Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Aplicación móvil para publicaciones dirigidas a entidades locales

Estudiante: Hugo Morales Martínez
Dirección: Diego Andrade Canosa

A Coruña, febreiro de 2021.

A mi familia y amigos.

Agradecimientos

Me gustaría agradecer en primer lugar a mis padres y a mis abuelos. Que me han dado el apoyo necesario para llegar hasta aquí.
A Estela, que de forma incondicional me ha ayudado sin saberlo y poco a poco se ha convertido en una pieza fundamental en mi vida.
A mis amigos, que me han dado el apoyo y han tirado de mí incluso en aquellos momentos que era más fácil tirar la toalla.
A mi tutor, Diego Andrade Canosa, que me ha ayudado a lo más difícil, tener una idea con la que empezar y me ha acompañado en la realización del proyecto.

Resumen

El objetivo de este trabajo fin de grado es la creación de una aplicación móvil destinada a recoger la información relativa a un determinado núcleo poblacional. La aplicación se construye para un territorio genérico siendo posible adaptar su uso a distintos núcleos. Los usuarios que usen esta herramienta pueden consultar las noticias de su población; un tablón de actividades deportivas, culturales y comunicados dentro de su zona; también disponen de un mapa interactivo, que les permite denunciar las deficiencias en su zona, desperfectos en el mobiliario y daños en la vía pública.

Para llevar a cabo la finalidad del proyecto, en primer lugar, se ha realizado un análisis previo para conocer en detalle los requisitos de la aplicación. Posteriormente se ha llevado a cabo el diseño, el desarrollo y las pruebas de las funcionalidades de la misma.

En el desarrollo de esta herramienta, se empleó el framework Flutter basado en el lenguaje Dart, y Firebase para el almacenamiento centralizado de los datos de la aplicación, ambas tecnologías desarrolladas por la compañía Google.

En algunas funcionalidades se obtienen datos de APIs de terceros: las noticias se obtienen a través de la API de NewsAPI y con la API de Google Maps se construye el mapa que se emplea en las incidencias.

Para gestionar la realización del proyecto se ha adaptado la metodología ágil Scrum, que permite dividir el desarrollo en Sprints.

Abstract

The objective of this final degree project is the creation of a mobile application destined to collect the information related to a certain population nucleus. The application is built generically, making it possible to adapt its use to different population centres. The users of this app can check the news of their population; a board of sports, cultural and communication activities within their area; they also have an interactive map, which allows reporting deficiencies in their area, damage to furniture, and damage to public roads.

To carry out the purpose of the project, in the first place, a preliminary analysis has been carried out to know in detail the application requirements. Then the design, development, and finally the testing of the application's functionalities.

For the development of this project, we use the Flutter framework based on the Dart language and Firebase as a database, both technologies developed by the Google Company. With Firebase, we performed the application's backend service, where all the data are stored.

Third-party APIs have been used too for some functionalities: the NewsAPI API has been

used to obtain news and Google Maps API has been used for the map.
For project management, we choose the agile Scrum methodology, which allows us to divide the development into sprints.

Palabras clave:

- API
- Dart
- Firebase
- Flutter
- Framework
- Scrum
- Modular
- Genericidad
- Sprints

Keywords:

- Dart
- Firebase
- Flutter
- Framework
- Scrum
- Modular
- Genericity
- Sprints

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Gracias a los dispositivos móviles, cuya utilización se ha vuelto común para la mayor parte de la población mundial, surge la posibilidad de poder crear herramientas con un mayor alcance, que nos faciliten la comunicación con nuestro entorno y nos mantengan informados de lo que pasa a nuestro alrededor. En este contexto, surge la idea de crear una aplicación con la que informarnos de lo que sucede o de los eventos que se van a producir en nuestro entorno y desde donde poder informar de aquello que no nos guste o esté mal en nuestra zona de una forma más directa. Actualmente solo existen algunas aplicaciones que permiten estas características en algunas capitales o grandes ciudades debido a su elevado coste económico.

El principal objetivo de este trabajo es la creación de una aplicación de bajo coste, amoldable a cualquier población, que permita una comunicación fluida entre la administración y sus vecinos. De este modo, no solamente se ayuda al usuario vecino a estar informado, sino que también se ayuda al gobierno de esa población a ser más interactivo y le permite conocer las quejas y opiniones de sus vecinos de una forma más directa.

Con la idea de llegar al mayor número de personas posible y de poder ser adaptada a cualquier entorno, es necesario crear una interfaz personalizable, intuitiva y cómoda para cualquier usuario.

1.1 Motivación

La idea de realizar el desarrollo de esta aplicación surge por la necesidad existente en muchas poblaciones de disponer de un medio o herramienta eficaz para informar a sus vecinos y desde donde poder obtener una mayor participación de ellos.

En poblaciones pequeñas no es fácil mantener medios de comunicación directa, por lo que muchas veces se carece de un medio exclusivo para una zona concreta. Con esta aplicación se pretende proporcionar un medio económico y capaz de llegar a muchos usuarios, a cualquier núcleo sin importar su tamaño. Gracias a que su gestión puede ser realizada por la propia administración local, se reducen los costes derivados de mantenimiento en comparación con otros medios (radio, televisión, prensa, etc.) haciendo que sea más sostenible.

Además, gracias al *framework* de desarrollo empleado se construye una solución multiplataforma, lo que facilita llegar a un mayor número de usuarios, sin importar el dispositivo que se emplee, mejorando su alcance sin necesidad de una inversión económica muy grande.

1.2 Objetivos

El objetivo principal de la aplicación es poner a disposición de los diferentes núcleos de población una herramienta sencilla, económica y útil que permita a cada vecino estar informado y le permita comunicarse con la administración de forma más fácil.

Se busca un comportamiento adaptativo en lo que a funcionalidades se refiere, pudiendo adaptarlas a los requisitos y necesidades del entorno donde se vaya a desplegar, añadiendo o eliminando características en función de las necesidades de la población. Sin olvidar los diferentes perfiles de usuarios que utilizarán la aplicación.

Para ello, la aplicación generada debe cumplir los siguientes requisitos:

- Funcionales:
  - Satisfacer las necesidades informativas de la población mediante la incorporación de un apartado donde poder consultar las noticias más relevantes y recientes de la zona.
  - Proporcionar al usuario un tablón de eventos, donde poder consultar todas las actividades que se organicen en su población y donde se mostrarán también anuncios de la propia administración. Proporcionando datos relativos a horarios y ubicaciones de las mismas.
    Dotar así a la administración de un medio por donde hacer llegar los eventos que se organicen y los comunicados oficiales a la población de forma directa.
  - Proporcionar un sistema de gestión de usuarios útil y eficaz, capaz de registrar nuevas altas, permitir acceso a los ya existentes a través de un correo y una contraseña o mediante un servicio de terceros. E incluso proporcionar el acceso de

forma anónima a algunas funcionalidades para aumentar su utilidad.

  - Proporcionar un medio por donde un usuario registrado comunique a su administración los desperfectos e incidencias que encuentre en su zona y desde donde pueda comprobar si han sido subsanadas o no.
    Proporcionar a la administración un mecanismo de inspección, donde poder ver qué desperfectos denuncian sus vecinos.

- No funcionales:

  - Debe proporcionarse una herramienta que pueda ser adaptada a cualquier entorno. Permitiendo modificar sus tipografías, colores e imágenes en función de la población que abarque.
  - Debe contar con una interfaz intuitiva y limpia, que haga atractivo y sencillo su uso para cualquier perfil de usuario. Además, deberá guiar al usuario en su manejo y advertir de los errores que pueda cometer.
  - La herramienta debe ser modular en lo que a funcionalidades se refiere, de modo que sea posible habilitar o deshabilitar características en función de las necesidades del cliente.
  - Deben proporcionarse medios de validación de datos de usuarios y la información relativa al usuario debe estar protegida.
  - La respuesta de la aplicación ha de ser lo más rápida posible incluso en condiciones desfavorables como puede ser la baja calidad en la conexión a internet.
  - La herramienta proporcionada debe ser escalable de modo que permita el incremento del número de usuarios y sea capaz de manejar un tráfico grande de usuarios.

1.3 Estructuración de la memoria

La estructura del resto de la memoria está compuesta de los siguientes capítulos:

- **Estado del arte**: Análisis de las aplicaciones similares existentes y de las características y requisitos que proporcionan.
- **Metodología y planificación**: Explicación del método de trabajo que se utiliza para llevar a cabo el desarrollo de la aplicación y su adaptación al proyecto. También se expone la planificación temporal y una estimación de los costes económicos derivados de su realización.

- **Análisis**: Se detallan los casos de uso necesarios y los requisitos funcionales y no funcionales que ha de cumplir la aplicación para ser adecuada. Se hace también una descripción de cada una de las tecnologías empleadas para cumplir esos requisitos.
- **Diseño e implementación**: Estudio relativo al diseño previo a la implementación, creando un modelo a seguir. Posteriormente en el mismo capítulo, se aborda una explicación sobre el proceso de implementación.
- **Pruebas**: Se explican las pruebas realizadas sobre la aplicación. Explicando en qué consiste cada una, cómo se realizó, los resultados obtenidos y las consecuencias que desencadenaron.
- **Conclusiones**: Capítulo final en el que se revisan los objetivos alcanzados en el proyecto y el posible trabajo futuro.

Capítulo 2

Estado del arte

En este capítulo se aborda, teniendo en cuenta los objetivos descritos en el capítulo anterior, una búsqueda de soluciones existentes en el mercado actual, con el objetivo de formar una perspectiva de lo que ofrecen y cómo aplicar y mejorar esas ideas en nuestra implementación.

En las siguientes secciones, se aborda una explicación de algunos ejemplos encontrados, diferenciándolos en función de si son aplicaciones **Ad-hoc**, tratados en el *apartado 2.2*, o no son **Ad-hoc**, explicados en el *apartado 2.1*. Esta distinción se basa en que hayan sido creados para una zona concreta y no se pueden aplicar a otras (**Ad-hoc**) o bien sean aplicaciones derivadas de productos genéricos, siendo de esta forma aplicables a cualquier población (no **Ad-hoc**). Se exponen todas las características ofertadas por los desarrolladores y se hace un análisis de las aplicaciones ya desplegadas, haciendo hincapié en las características comunes, ya que nos sirven para denotar las funcionalidades fundamentales. Finalmente en la *sección 2.3*, se explican las conclusiones del estudio realizado.

Existen algunos ejemplos de aplicaciones genéricas adaptables a diferentes núcleos poblacionales que cubren los objetivos que nos hemos fijado. Nos centraremos en las ofrecidas por *Estudio Alfa* [1] y *Gestión Ayuntamiento* [2], ambas ofrecen el despliegue de una aplicación adaptada a un núcleo de población determinado. A través de ambas herramientas se puede mantener informados a los habitantes, permitir la realización de gestiones, comunicar incidentes que ocurran en la vía pública, promocionar productos de comercio local e informar sobre los puntos turísticos de referencia de la zona.

Existen también algunas aplicaciones no adaptables que cubren funcionalidades para una determinada población. Es el caso, por ejemplo, de las aplicaciones *"Avisos Madrid"* y *"AppValència"* que son productos generados especialmente para estas ciudades. Estas dos aplicaciones

ofrecen unas características similares a las ofertadas por las empresas antes mencionadas.

En las aplicaciones anteriormente citadas, se observan unas características comunes, que pueden ser consideradas como las mínimas funcionalidades necesarias para que la aplicación sea útil.

2.1 Análisis de aplicaciones genéricas existentes

Las empresas *Estudio Alfa* y *Gestión Ayuntamiento* ofrecen soluciones web y móviles destinadas a resolver diferentes problemas. En esta oferta encontramos sendas aplicaciones móviles para ayuntamientos.

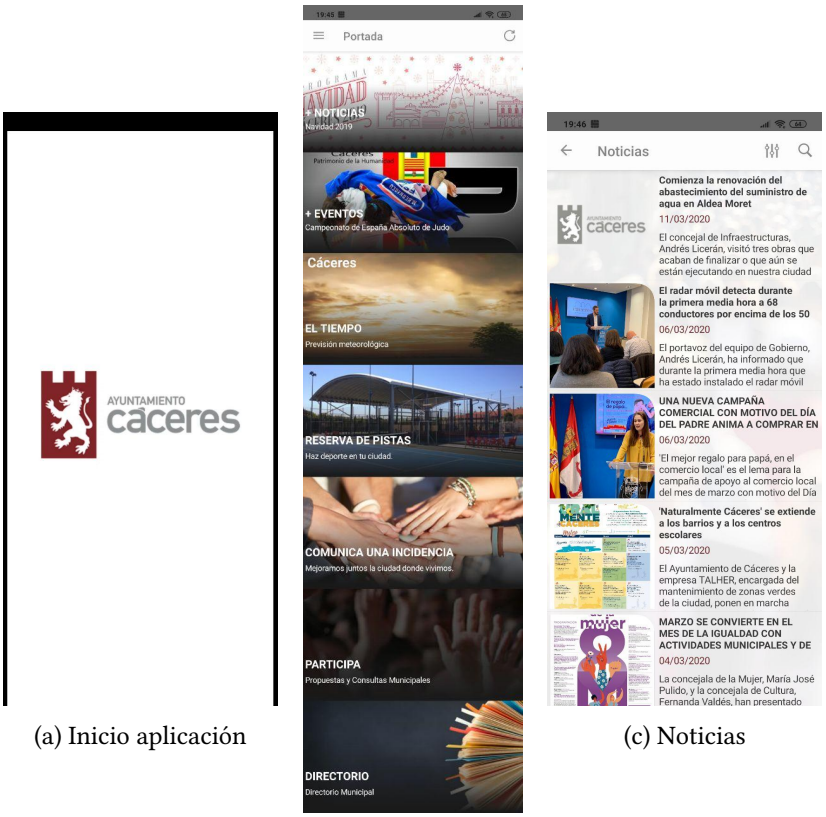
Estas aplicaciones se ofrecen con el objetivo de proporcionar una solución capaz de potenciar la economía de la localidad y ofrecer una mejor comunicación con los ciudadanos. Para ello ambas aplicaciones implementan los siguientes módulos:

- **Informe de daños**: donde los usuarios pueden informar sobre las incidencias que ocurren en el municipio, incluyendo una descripción, fotografías y una ubicación **GPS**. Esta información se almacena en una base de datos y es comunicada al gestor.
- **Enlaces de interés**: Apartado donde se incluye información relevante para la comarca (horarios de autobuses, tiempo, farmacias de guardia, etc.).
- **Noticias**: Conexión con la página del propio ayuntamiento para extraer y ofrecer las noticias del municipio.
- **Programa de fiestas**: Se muestran los carteles de las fiestas del municipio.
- **Encuestas**: Para fomentar la participación de los vecinos, se podrán realizar encuestas para recibir los comentarios de los usuarios, facilitando la toma de decisiones.

Estas empresas proporcionan unas aplicaciones flexibles y adaptables a las necesidades del usuario final, pudiendo habilitar o deshabilitar funcionalidades a petición del cliente. Ofrecen una solución nativa multiplataforma para iOS y Android. Empleando un gestor de bases de datos **MySQL** de la compañía Oracle para almacenar los datos.

Se ofrece un plan de formación para aquellas personas que vayan a ser las encargadas de administrar la aplicación, donde se les enseña a usar un portal web proporcionado para la gestión de la aplicación.

Para comprobar su funcionamiento, se han hecho pruebas sobre despliegues de estas empresas, en concreto se ha observado la aplicación para el ayuntamiento de Cáceres. [3].



(a) Inicio aplicación

(b) Portada

(c) Noticias

Figura 2.1: Ventanas de la aplicación del Ayuntamiento de Cáceres (Inicio, portada, noticias).

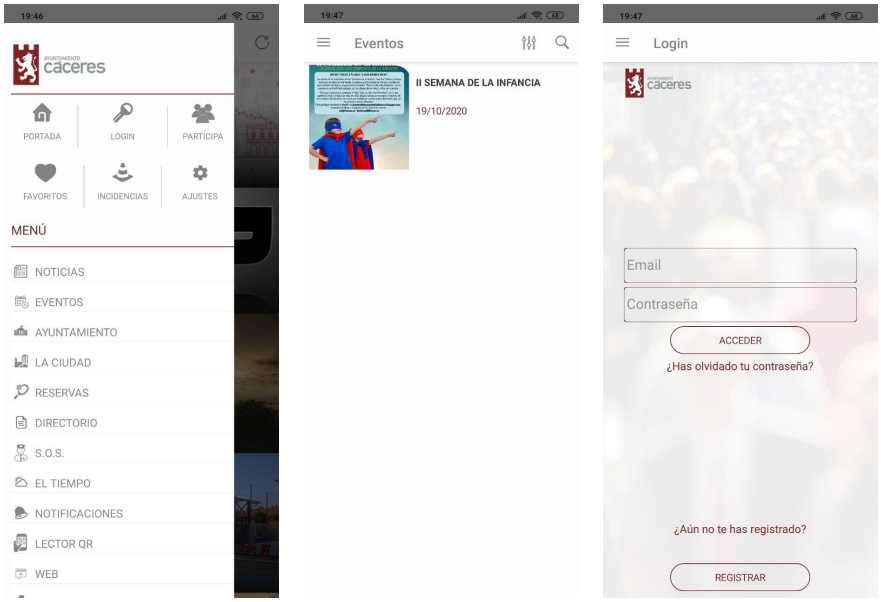
En el inicio de la aplicación, reflejado en *figura 2.1a*, se muestra el escudo corporativo del ayuntamiento al que pertenece, esta es una de las opciones de personalización con la que cuenta la app. Una vez que accedemos a la aplicación, nos encontramos con el menú principal, donde hay una lista de las informaciones que podemos consultar, como se ve reflejado en la *figura 2.1b*:

- **Noticias**: Se muestra un listado, como se ve en la *figura 2.1c*, de las últimas noticias del ayuntamiento, donde nos permite interaccionar con ellas (llevándonos al navegador), filtrarlas por fecha o buscar noticias concretas.

- **Eventos**: Nos muestra un listado similar al de las noticias como podemos ver en la *figura 2.2b*, permitiéndonos las mismas acciones que para las noticias: filtrar, buscar o interaccionar con ellas.

- **El tiempo**: Muestra una tarjeta con el tiempo en ese momento, y si interaccionamos con ella nos lleva a la página del ayuntamiento donde aparece más información.
- **Identificación de usuarios**: Mediante el registro (*figura 2.2c*) o el inicio de sesión, permite realizar reservas de las pistas de pádel municipales, comunicar incidencias o participar en encuestas.
- **Comunica una incidencia**: Mediante el registro permite añadir una incidencia en el municipio (Esto no ha sido posible probarlo, porque todavía no está habilitado).
- **Participa**: Una vez registrados, permite acceder a una lista de encuestas en relación con acontecimientos surgidos en el día a día del ayuntamiento.
- **Directorio**: En este apartado se muestra una lista con los teléfonos de todas las instituciones públicas dentro de este ayuntamiento (Oficina de servicios urbanos, secretaría general, universidad popular, etcétera).

La aplicación cuenta además con un menú lateral (*figura 2.2a*), desde donde se tiene un acceso rápido a todas las funcionalidades que hay en portada y desde donde podemos realizar el inicio de sesión y configurar las preferencias del usuario.



(a) Menú lateral (b) Eventos (c) Inicio de sesión

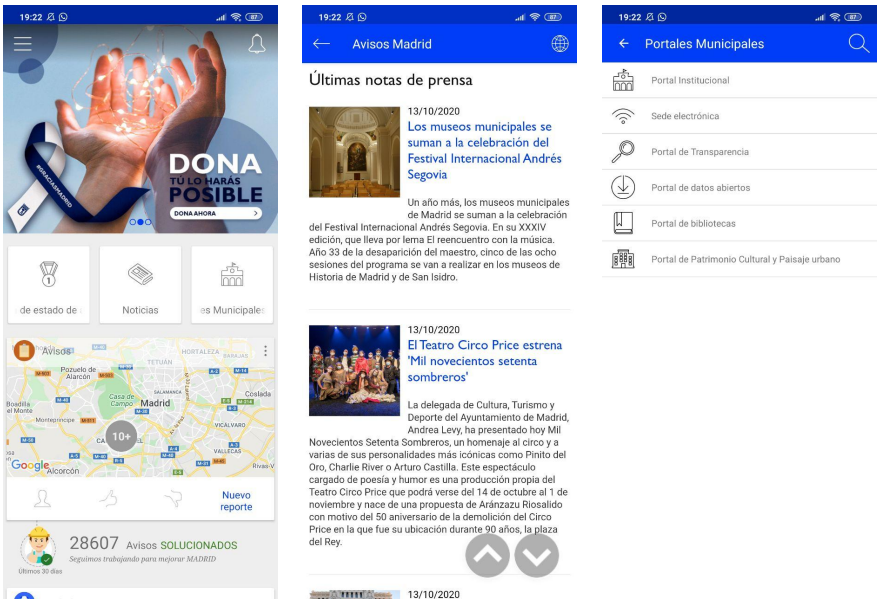
Figura 2.2: Ventanas de la aplicación del Ayuntamiento de Cáceres (Menú, eventos, inicio de sesión).

2.2 Análisis de aplicaciones Ad-hoc

En esta sección nos vamos a centrar en el análisis de dos aplicaciones **Ad-hoc**, es decir, se han creado expresamente para dos ciudades concretas (Madrid y Valencia) y no es posible adaptarlas a otros núcleos. Ambas aplicaciones cuentan con una estructura similar y unas características comunes muy definidas. Se componen de los siguientes apartados:

- Noticias
- Eventos
- Incidencias
- Enlaces de interés

Como en el apartado anterior, para poder comprobar mejor su funcionamiento y las opciones que ofrecen, se ha estudiado una de ellas, en concreto la app *Avisos Madrid* [4]:



(a) Portada (b) Noticias (c) Portal municipal

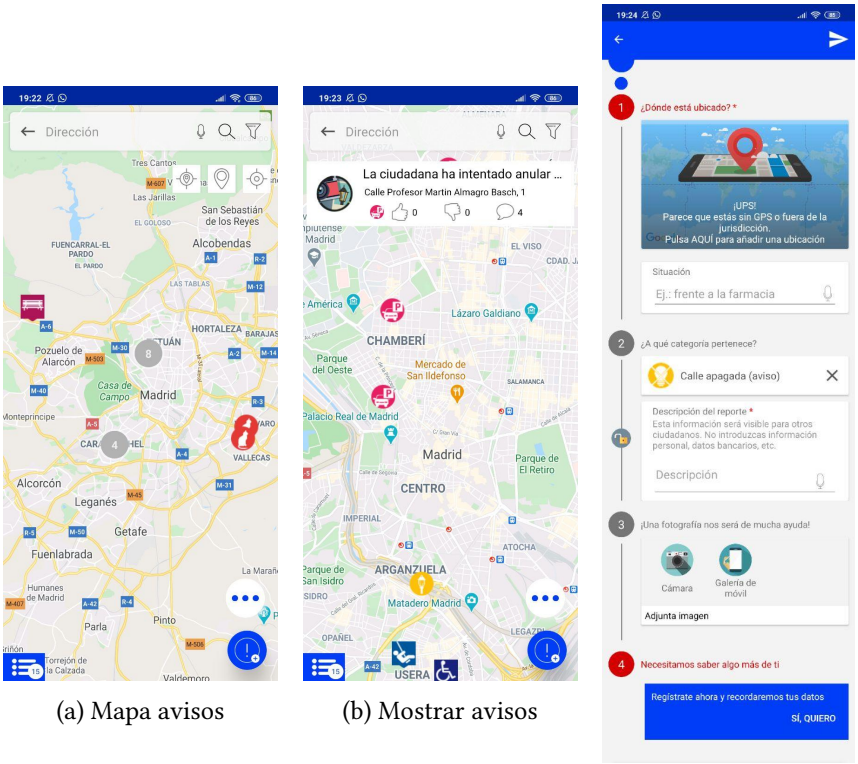
Figura 2.3: Ventanas de la aplicación Avisos Madrid (Portada, noticias, portal municipal).

En este caso, en el inicio de la aplicación no se muestra ninguna imagen corporativa, ya que se abre directamente la portada (*figura 2.3a*). Donde aparecen unos accesos rápidos a las funciones principales:

- **Noticias**: Ventana en la que se muestra un listado de las últimas noticias del ayuntamiento de Madrid (*figura 2.3b*). En esta ventana se pueden buscar noticias, filtrar por

fecha e interaccionar con ellas abriéndolas en el navegador o compartiéndolas en redes sociales.

- **Eventos**: No disponen de una ventana propia, son desplegados en la propia portada de la aplicación, como se puede ver en la *figura 2.3a*, donde aparecen los eventos y anuncios en la parte superior.
- **Portal Municipal**: En esta vista, correspondiente a la *figura 2.3c*, se muestra una lista con opciones sobre informaciones de la administración pública del ayuntamiento y enlaces de interés sobre el mismo. Estas opciones se enlazan a la página del ayuntamiento donde son tratados cada uno de esos temas.
- **Incidencias**: A esta ventana accederemos a través del mapa que aparece en la portada de la aplicación. Se abre un mapa en la pantalla (*figura 2.4b*), donde podremos ver las incidencias denunciadas por otros usuarios como en la *figura 2.4a* y desde donde podemos denunciar las nuestras a través de un formulario (*figura 2.4c*).



(a) Mapa avisos (b) Mostrar avisos (c) Formulario avisos

Figura 2.4: Ventanas de la aplicación Avisos Madrid (Mapa avisos, avisos, formulario avisos).

2.3 Conclusiones obtenidas

Tras la observación de estas aplicaciones se elaboraron las siguientes conclusiones:

- Debe ser intuitiva y tener un diseño sencillo. En alguna de las aplicaciones analizadas resulta complicado acceder a algunas características por culpa de su diseño o por falta de indicaciones.
- Mostrar solamente la información necesaria. En algunos casos, se expone demasiada información, haciendo que se vea una interfaz poco clara.
- Una respuesta rápida a las acciones del usuario es fundamental. En las aplicaciones genéricas, la respuesta a eventos es muy lenta y produce muchos errores. La agilidad en la interacción del usuario hace que la experiencia de uso sea mucho más atractiva.
- Todas cuentan con unos apartados comunes: Una sección de noticias de la zona, otra de eventos donde se recogen actividades (deportivas, culturales, fiestas, etc.), un apartado con información sobre el ayuntamiento y un formulario donde denunciar incidencias.

Capítulo 3

Metodología y planificación

En este capítulo se explica la metodología escogida para el proceso de desarrollo y las herramientas empleadas para su gestión. En el proyecto se ha seguido la metodología ágil Scrum. Organizando el proyecto en *Sprints*, que sirven de guía para organizar e implementar la aplicación de forma más ágil y sencilla.

En la *sección 3.1* se hace una introducción a la metodología utilizada. Una vez definidas las características base de la metodología, en la *sección 3.2* se detalla cómo se han adaptado esas características en beneficio del desarrollo del proyecto.

En la *sección 3.3*, se habla de la herramienta empleada para hacer la gestión y planificación de la metodología sobre el proyecto. Finalmente, en la *sección 3.4* se presenta una planificación de costes previa a la implementación.

3.1 Metodología Scrum

Scrum es una de las metodologías ágiles más empleadas en los últimos tiempos, es utilizada para mitigar los riesgos que se puedan producir durante la realización de un proyecto, ya que sirve para fijar unas pautas a seguir y permite trabajar más fácilmente con proyectos con múltiples cambios. Proporcionándonos una mayor flexibilidad frente a otras metodologías convencionales.

Permite la coordinación tanto de grupos reducidos como de grupos grandes, pudiendo abarcar proyectos de cualquier tamaño sin que suponga un problema; es por eso que se está exportando a otros ámbitos diferentes al desarrollo software.

Esta metodología se aplica organizando el proyecto en *Sprints*. Estos *Sprints* son entregas

parciales del producto final, lo cual permite la realización de proyectos con fecha límite, ya que con Scrum se marca un seguimiento diario con revisiones. Gracias a esta técnica se aumenta la productividad, la calidad del producto y se mejora el seguimiento del proyecto.

En las siguientes secciones se explicarán los roles de cada integrante del proyecto, los documentos necesarios y las pautas a seguir según Scrum. A continuación se puede ver un esquema del funcionamiento de la metodología Scrum y de los roles que intervienen.

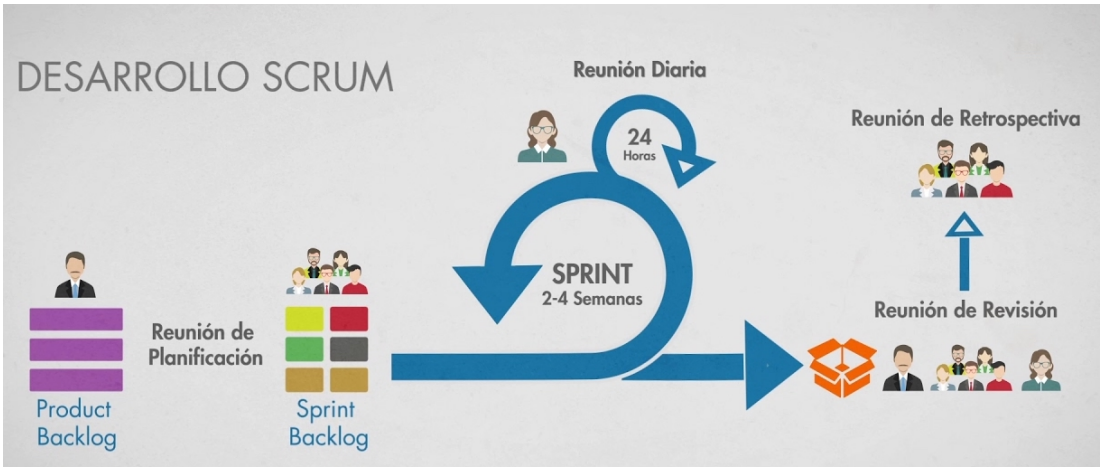


Figura 3.1: Esquema de los componentes de la metodología Scrum

3.1.1 Roles

A cada integrante del grupo se le asigna un rol que cumplir en el desarrollo:

- **Product Owner**: Persona encargada de representar al cliente o clientes del proyecto, se encarga de gestionar el *product backlog*, donde se definen las prioridades y el contenido de los *Sprints* para optimizar los resultados.
  Debe ser una persona que conozca el proyecto en profundidad, ya que todas las decisiones o cambios en el proceso de desarrollo deben pasar primero por sus manos, es por eso que el éxito depende en gran parte de su trabajo.

- **Scrum Master**: Es la persona encargada de gestionar el proyecto, pero no ha de confundirse con un jefe jerárquico o una autoridad. Su función es velar por el cumplimiento de la metodología y estar a disposición del equipo eliminando cualquier impedimento que puedan tener o quitando tareas innecesarias. Deberá participar en las reuniones, explicar la metodología al equipo y al *product owner*. Asegurándose de que se cumplan los objetivos temporales marcados.

- **Equipo Scrum o Equipo de desarrollo**: Conjunto de integrantes del grupo con la cualificación necesaria para llevar a cabo los *Sprints* del proyecto. Este grupo es independiente al resto de roles y se organizan entre ellos, no existe una jerarquía en él. Deberá ser de un tamaño suficiente para poder llevar a cabo todas las tareas en el tiempo estipulado y no demasiado grande para evitar entorpecimiento y ralentizaciones.

3.1.2 Eventos Scrum

Sirven para llevar un seguimiento del desarrollo, estos eventos se producen antes, durante y después de cada iteración. Su duración y organización es preestablecida, lo cual mejora la eficiencia del proceso y aumenta la productividad:

- **Sprint**: En el marco de desarrollo Scrum, se refiere a cada una de las iteraciones en las que se divide el proyecto. Con una duración estipulada, de entre una semana y unos meses, permite alcanzar objetivos concretos en un tiempo predefinido, así como llevar un ritmo constante de trabajo. Cada Sprint en los que se divida el proyecto ha de tener una complejidad y duración similar, para mantener un mismo ritmo de trabajo. Una vez terminado cada Sprint, el producto generado deberá ser totalmente funcional y pasará los test prefijados, pudiendo dar comienzo al siguiente *Sprint*.

- **Planificación del Sprint**: Reunión del equipo de Scrum que se produce antes del comienzo de cada *Sprint*. Donde se escoge cuál será el siguiente Sprint a realizar, basándose en el resultado obtenido en el Sprint anterior. Se hará una planificación temporal del *Sprint* a realizar y se marcará el objetivo principal a alcanzar y los medios para ello, esta reunión tiene una duración prefijada, no más de 8 horas.

- **Scrum diario**: Cada día, con una duración corta, el equipo Scrum se reúne para poner en conocimiento del resto del equipo el trabajo a realizar en la próxima jornada. Sirviendo para contextualizar al equipo en la situación actual del *Sprint*.
  Estas juntas mejoran la coordinación y comunicación del equipo, mejorando el flujo de trabajo y facilitando el cumplimiento de objetivos.

- **Revisión del Sprint**: Aunque puede integrarse en las reuniones de planificación del Sprint, se suelen realizar al final de cada iteración. Sirven para que el equipo de desarrollo comunique los casos de uso que han implementado, los cambios que han introducido, los problemas que han surgido y cómo los han solventado. En ellas, el *product owner* comprueba qué objetivos se han completado y cuáles no, e introducirá nuevos plazos y cambios basándose en los progresos realizados y a los costes obtenidos.

- **Retrospectiva del Sprint**: Al final de cada Sprint suele realizarse una reunión más aparte de la destinada a la revisión del Sprint. Donde se debatirá sobre los objetivos alcanzados, qué se puede hacer para mejorar posteriores *Sprints* y cómo afectará al equipo. Una vez finalizada se considera que el *Sprint* ha acabado y podrá continuarse con el siguiente, repitiendo el mismo proceso.

3.1.3 Artefactos y documentación

En Scrum existen unos documentos encargados de proporcionar un nivel de transparencia plena a todos los integrantes del equipo de desarrollo, estos documentos son conocidos como artefactos, y en ellos se dispone la información necesaria para que todos los miembros del equipo cuenten con la misma información, permitiéndoles comprender mejor las circunstancias del proyecto.

- **Product backlog**: Documento que contiene un listado de todos los requisitos que ha de cumplir el proyecto, este listado se genera a raíz del trabajo del *product Owner* con el cliente. De tamaño variable y modificado a lo largo del proceso de desarrollo, añadiendo o eliminando elementos en función de las necesidades del proyecto.
  Todos los integrantes pueden consultar este documento, pero solo puede ser editado por el *product owner*.

- **Sprint backlog**: Documento en el que se recoge el trabajo que se va a realizar en un Sprint, los objetivos a alcanzar y el planning para lograrlo. Este documento se deriva de la información contenida en el *product backlog*.
  Es administrado por el equipo Scrum, encargándose de modificarlo si fuese necesario.

- **Incremento**: Puntos completados del *backlog* en un Sprint y el resultado de los incrementos de los *Sprints* anteriores. Debe cumplir los estándares de calidad impuestos y ser funcional, aunque no se lance el producto.

- **Épicas**: Cantidades de trabajo que no pueden ser realizadas en una misma iteración, obligando a su desglose en subtareas midiendo su tamaño mínimo con lo que se conoce como historia de usuario. Con las épicas podemos mostrar el estado de las grandes funcionalidades de la aplicación.

- **Historias de usuario**: Descripciones de los casos de uso y funcionalidades requeridas, proporcionadas por el cliente. Ayudan a comprender mejor el resultado esperado.

3.2 Aplicación de Scrum al proyecto

Para realizar este proyecto se ha empleado una versión muy simplificada de la metodología Scrum, explicada en la sección anterior, haciendo algunas modificaciones para que la parte de desarrollo pueda realizarse por una sola persona, ya que normalmente el equipo de desarrollo está formado por un equipo de al menos dos personas; así mismo le serán asignados varios roles de gestión al tutor del proyecto. Asignación de roles:

- **Product Owner**: Diego Andrade Canosa
- **Scrum Master**: Diego Andrade Canosa
- **Equipo Scrum**:
  - **Analista**: Hugo Morales Martínez
  - **Diseñador**: Hugo Morales Martínez
  - **Programador**: Hugo Morales Martínez

En la planificación inicial del proyecto se establecieron los siguientes *Sprints* para la realización del proyecto:

Sprint 1

- **Duración estimada**: Una semana
- **Descripción de objetivos**:
  1. Obtención de los conocimientos necesarios para la utilización de las tecnologías involucradas en el proyecto (Flutter, Dart y Firebase).
  2. Análisis de requisitos funcionales, no funcionales y definición de los casos de uso.
  3. Definición de la estructura general de la aplicación mediante diagramas de flujo y *MockUps*.
  4. Creación de la estructura del proyecto.

Sprint 2

- **Duración estimada**: Una semana
- **Descripción de objetivos**:
  1. Creación de la base de datos en Firebase para almacenar los usuarios.
  2. Creación de la base de datos para almacenar eventos e incidencias.

Sprint 3

- **Duración estimada**: Dos semanas
- **Descripción de objetivos**:
  1. Creación del entorno de registro e inicio de sesión del usuario.
  2. Integración de Firebase Authentication para el registro y login de usuarios.
  3. Implementación de registro y login con Google.
  4. Recuperación de contraseña mediante el envío de correo personalizado.

Sprint 4

- **Duración estimada**: Dos semanas
- **Descripción de objetivos**:
  1. Primera versión del apartado de noticias de la aplicación.
  2. Integración de la **API** de noticias (*NewsAPI*).

Sprint 5

- **Duración estimada**: Dos semanas
- **Descripción de objetivos**:
  1. Primera versión del apartado tablón de la aplicación.
  2. Integración de la base de datos de Firebase de poblaciones para almacenar eventos por tipo.

Sprint 6

- **Duración estimada**: Dos semanas
- **Descripción de objetivos**:
  1. Primera versión del apartado de añadir incidencias de la aplicación.
  2. Integración de la **API** de *Google Maps*.
  3. Integración de la base de datos de Firebase para el almacenamiento de incidencias.
  4. Implementación del servicio de geolocalización.

Sprint 7

- **Duración estimada**: Una semana
- **Descripción de objetivos**:
  1. Implementación del *drawer* de información del usuario.

2. Integración de la base de datos de usuario en el *drawer* para mostrar la información del usuario.
3. Integración de la base de datos de incidencias para mostrar las incidencias del usuario en su *drawer* y su estado.

Sprint 8

- **Duración estimada**: Una semana
- **Descripción de objetivos**:
  1. Retoques estéticos para la adaptación a *Material Design*.
  2. Retoques para ayudar a la accesibilidad de la aplicación.
  3. Retoques derivados de la experiencia de uso de los usuarios de prueba.

Documentación

- **Duración estimada**: Dos semanas
- **Descripción de objetivos**:
  1. Redacción y recopilación de la documentación relativa al proyecto.

Reunión final

- **Duración estimada**: Un día
- **Descripción de objetivos**:
  1. Revisión del último *Sprint*.
  2. Revisión de la documentación generada.

3.3 Utilización de Trello para la gestión de iteraciones

Trello [5] es una aplicación de gestión de proyectos, empleada para facilitar la coordinación de equipos y la organización de tareas. Es el software escogido para administrar el proyecto.

Con Trello mantendremos el documento *"product backlog"*. Desde esta plataforma se puede hacer un seguimiento de las tareas involucradas en cada iteración. Para ello, en el tablón se han creado varias listas donde se insertan las tareas en función de su estado.

Esta herramienta como se presenta es un entorno web, resulta muy útil para realizar trabajos en equipo, todos pueden acceder al mismo tablón y ver el estado en cualquier momento. Además gracias a las anotaciones es fácil asignar una tarea determinada a un integrante del grupo y que este no se encuentre perdido en su desarrollo.

A continuación en la *figura 3.2*, y a modo de ejemplo, se puede ver: la gestión de la iteración número tres desde Trello, la organización de las tareas basándose en su estado y cómo se han ilustrado mediante imágenes. Estas imágenes hacen más sencillo identificar en qué momento se produjo un cambio determinado.

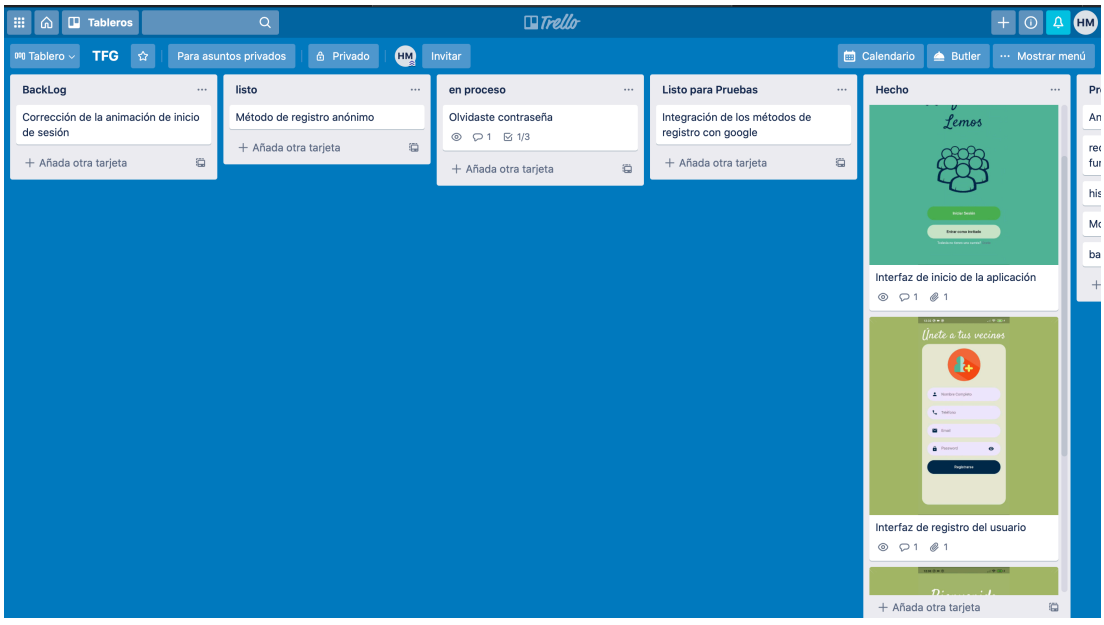


Figura 3.2: Seguimiento de la iteración 3 con Trello

Aunque a simple vista se vean tareas con poco nivel de detalle, dentro de cada uno de los elementos de las listas, se detallan las subtareas que hay implícitas en cada una de ellas y un

porcentaje de compleción de la tarea que las engloba, también nos permite añadir imágenes que facilitan la comprensión del estado actual, y comentarios que pueden ser empleados como recordatorios o para informar a los demás integrantes del grupo del estado actual de la tarea.

También a modo de ejemplo, en la *figura 3.3* se puede ver la información que hay dentro de la tarea *"Olvidaste contraseña de la figura anterior"* (*figura 3.2*).

Figura 3.3: Descripción dentro de cada tarea

3.4 Costes

Para poder realizar una estimación de los costes del desarrollo del proyecto, es necesario tener en cuenta el salario de todos los implicados en el desarrollo y los costes de los recursos materiales y software empleados.

Para ello, una vez que se han definido las tareas de cada Sprint, se ha hecho un *diagrama de Gantt* estimando la duración de cada *Sprint*, asignando los recursos para cada tarea, y en función del coste de **€/hora** por hombre se ha obtenido una estimación de coste por tarea, por Sprint y por el proyecto en total. A continuación se puede ver una tabla donde se reflejan todos estos datos:

	Nombre de tarea	Duración	Comienzo	Fin	Predecesor	Nombres de los recursos	Costo
1	**Aplicación Publicaciones en Entidades Locales**	**80 días**	**lun 01/06/20**	**vie 18/09/20**			**13.009,50 €**
2	**Iteración 1**	**6 días**	**lun 01/06/20**	**lun 08/06/20**		**Analista;Diseñador;Programador;Scrum Master**	**1.155,00 €**
3	Reunion de planificación	1 día	lun 01/06/20	lun 01/06/20		Analista, diseñador,programador, Scrum Master	514,50 €
4	Análisis	1 día	mar 02/06/20	mar 02/06/20	3	Analista;Scrum Master	143,50 €
5	Diseño	1 día	mié 03/06/20	mié 03/06/20	4	Diseñador	91,00 €
6	Implementacion	2 días	jue 04/06/20	vie 05/06/20	5	Programador	210,00 €
7	Tests y solución de errores	1 día	lun 08/06/20	lun 08/06/20	6	Diseñador;Programador	196,00 €
8	Finalización iteración 1	0 días	mar 09/06/20	mar 09/06/20	7		0,00 €
9	**Iteración 2**	**6 días**	**mar 09/06/20**	**mar 16/06/20**		**Analista, diseñador,programador, Scrum Master**	**1.155,00 €**
10	Reunión para planificacion y revisión	1 día	mar 09/06/20	mar 09/06/20		Analista, diseñador,programador, Scrum Master	514,50 €
11	Análisis	1 día	mié 10/06/20	mié 10/06/20	10	Analista	143,50 €
12	Diseño	1 día	jue 11/06/20	jue 11/06/20	11	Diseñador	91,00 €
13	Implementacion	2 días	vie 12/06/20	lun 15/06/20	12	Programador	210,00 €
14	Tests y solución de errores	1 día	mar 16/06/20	mar 16/06/20	13	Diseñador;Programador	196,00 €
15	Finalización iteración 2	0 días	mié 17/06/20	mié 17/06/20	14		0,00 €
16	**Iteración 3**	**10 días**	**mié 17/06/20**	**mar 30/06/20**		**Analista, diseñador,programador, Scrum Master**	**1.575,00 €**
17	Reunión para planificacion y revisión	1 día	mié 17/06/20	mié 17/06/20		Analista, diseñador,programador, Scrum Master	514,50 €
18	Análisis	1 día	jue 18/06/20	jue 18/06/20	17	Analista	143,50 €
19	Diseño	1 día	vie 19/06/20	vie 19/06/20	18	Diseñador	91,00 €
20	Implementacion	6 días	lun 22/06/20	lun 29/06/20	19	Programador	630,00 €
21	Tests y solución de errores	1 día	mar 30/06/20	mar 30/06/20	20	Diseñador;Programador	196,00 €
22	Finalización iteración 3	0 días	mié 01/07/20	mié 01/07/20	21		0,00 €
23	**Iteración 4**	**10 días**	**mié 01/07/20**	**mar 14/07/20**	**22**	**Analista, diseñador,programador, Scrum Master**	**1.575,00 €**
24	Reunión para planificacion y revisión	1 día	mié 01/07/20	mié 01/07/20		Analista, diseñador,programador, Scrum Master	514,50 €
25	Análisis	1 día	jue 02/07/20	jue 02/07/20	24	Analista	143,50 €
26	Diseño	1 día	vie 03/07/20	vie 03/07/20	25	Diseñador	91,00 €
27	Implementacion	6 días	lun 06/07/20	lun 13/07/20	26	Programador	630,00 €
28	Tests y solución de errores	1 día	mar 14/07/20	mar 14/07/20	27	Diseñador;Programador	196,00 €
29	Finalización iteración 4	0 días	mié 15/07/20	mié 15/07/20	28		0,00 €
30	**Iteración 5**	**10 días**	**mié 15/07/20**	**mar 28/07/20**	**29**	**Analista, diseñador,programador, Scrum Master**	**1.575,00 €**
31	Reunión para planificacion y revisión	1 día	mié 15/07/20	mié 15/07/20		Analista, diseñador,programador, Scrum Master	514,50 €
32	Análisis	1 día	jue 16/07/20	jue 16/07/20	31	Analista	143,50 €
33	Diseño	1 día	vie 17/07/20	vie 17/07/20	32	Diseñador	91,00 €
34	Implementacion	6 días	lun 20/07/20	lun 27/07/20	33	Programador	630,00 €
35	Tests y solución de errores	1 día	mar 28/07/20	mar 28/07/20	34	Diseñador;Programador	196,00 €
36	Finalización iteración 5	0 días	mié 29/07/20	mié 29/07/20	35		0,00 €
37	**Iteración 6**	**10 días**	**mié 29/07/20**	**mar 11/08/20**	**36**	**Analista, diseñador,programador, Scrum Master**	**1.575,00 €**
38	Reunión para planificacion y revisión	1 día	mié 29/07/20	mié 29/07/20		Analista, diseñador,programador, Scrum Master	514,50 €
39	Análisis	1 día	jue 30/07/20	jue 30/07/20	38	Analista	143,50 €
40	Diseño	1 día	vie 31/07/20	vie 31/07/20	39	Diseñador	91,00 €
41	Implementacion	6 días	lun 03/08/20	lun 10/08/20	40	Programador	630,00 €
42	Tests y solución de errores	1 día	mar 11/08/20	mar 11/08/20	41	Diseñador;Programador	196,00 €
43	Finalización iteración 6	0 días	mié 12/08/20	mié 12/08/20	42		0,00 €
44	**Iteración 7**	**6 días**	**mié 12/08/20**	**mié 19/08/20**	**43**	**Analista, diseñador,programador, Scrum Master**	**1.155,00 €**
45	Reunión para planificacion y revisión	1 día	mié 12/08/20	mié 12/08/20		Analista, diseñador,programador, Scrum Master	514,50 €
46	Análisis	1 día	jue 13/08/20	jue 13/08/20	45	Analista	143,50 €
47	Diseño	1 día	vie 14/08/20	vie 14/08/20	46	Diseñador	91,00 €
48	Implementacion	2 días	lun 17/08/20	mar 18/08/20	47	Programador	210,00 €
49	Tests y solución de errores	1 día	mié 19/08/20	mié 19/08/20	48	Diseñador;Programador	196,00 €
50	Finalización iteración 7	0 días	jue 20/08/20	jue 20/08/20	49		0,00 €
51	**Iteración 8**	**6 días**	**jue 20/08/20**	**jue 27/08/20**	**50**	**Analista, diseñador,programador, Scrum Master**	**1.155,00 €**
52	Reunión para planificacion y revisión	1 día	jue 20/08/20	jue 20/08/20		Analista, diseñador,programador, Scrum Master	514,50 €
53	Análisis	1 día	vie 21/08/20	vie 21/08/20	52	Analista	143,50 €
54	Diseño	1 día	lun 24/08/20	lun 24/08/20	53	Diseñador	91,00 €
55	Implementacion	2 días	mar 25/08/20	mié 26/08/20	54	Programador	210,00 €
56	Tests y solución de errores	1 día	jue 27/08/20	jue 27/08/20	55	Diseñador;Programador	196,00 €
57	Finalización iteración 8	0 días	vie 28/08/20	vie 28/08/20	56		0,00 €
58	**Documentación**	15 días	vie 28/08/20	jue 17/09/20	57	Programador	1.575,00 €
59	**Reunión Final**	1 día	vie 18/09/20	vie 18/09/20	58	Analista, diseñador,programador, Scrum Master	514,50 €

Figura 3.4: Costes en base a la planificación

En la *figura 3.5* podemos ver el *diagrama de Gantt*, en el que se recogen las tareas realizadas por *Sprint*, los recursos asignados y la duración estimada de cada una.

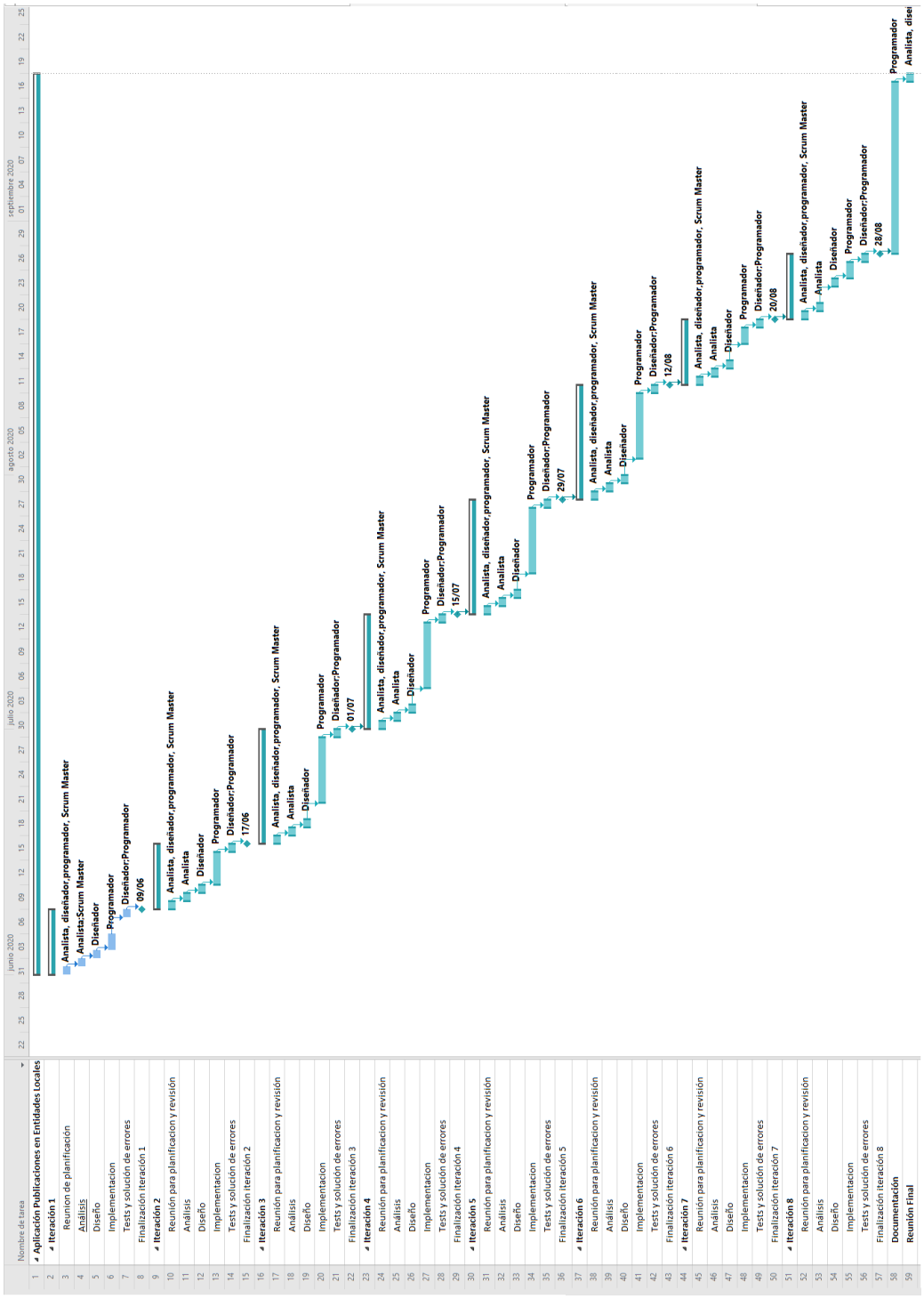


Figura 3.5: Diagrama de Gantt de la planificación

Costes recursos humanos:

Los costes reflejados en las *figura 3.4* y *3.5* son estimados tomando como base los datos sobre ofertas de empleo para los puestos de **Scrum Master** [6], **analista** [7], **diseñador** [8] y **programador** [9] en España. Tomando como jornada laboral 8 horas y correspondiendo a cada empleado el siguiente sueldo:

Puesto	**Sueldo medio anual**	**Euros/hora**
Scrum master	38.500 €	21,87
Analista	31.570 €	17,93
Diseñador	20.020 €	11,37
Programador	23.100 €	13,12

Tabla 3.1: Tabla de costes de los recursos humanos.

Costes recursos materiales y software:

Los costes materiales serán especificados mediante el valor de mercado de cada uno de los elementos empleados para el desarrollo. Siendo:

Recurso	**Coste**
Apple MacBook Pro	1.500 €
Xiaomi Mi9	449 €
Api Google Maps	0 €
NewsAPI	0 €
Figma	0 €
Trello	0 €
Microsoft Project	0 €
FireBase	0 €
Total	1.949 €

Tabla 3.2: Tabla de costes de los recursos materiales y software.

Atendiendo a los costes reflejados en las tablas anteriores, el coste estimado del proyecto asciende a los **14.958,5 €**.

Recurso	**Coste**
Recursos humanos	13.009,5 € €
Recursos materiales y software	1.949 €
Total	14.958,5 €

Tabla 3.3: Tabla de costes totales.

Capítulo 4

Análisis

En este capítulo se explican los requisitos que debe cumplir la aplicación, los diferentes actores que interactuarán con ella, los casos de uso que ha de satisfacer y se definen unos prototipos previos a la implementación.

En la *sección 4.1* se describen los usuarios que tendrán contacto con nuestra aplicación. Esto es importante, ya que basándonos en sus perfiles definiremos los requisitos y casos de uso que debemos satisfacer con nuestra aplicación.

En las secciones posteriores y basándonos en las necesidades de los perfiles definidos en la primera sección, se hace: Una descripción de los requisitos en la *sección 4.2*, clasificándolos en función de si son funcionales o no, en la sección posterior, la *4.3*, se describen también los casos de uso que debemos implementar para que la aplicación sea útil para estos perfiles.

Una vez se ha hecho la definición de los usuarios, los requisitos y los casos de uso, se realiza un diagrama de flujo (correspondiente a la *sección 4.4*) donde se define la secuencia a seguir para realizar cada caso y un esquema de datos de la aplicación (*sección 4.5*), donde se muestra una estructura de la información que almacena la aplicación. Posteriormente, en la *sección 4.6*, se hace un prototipo de la interfaz en la que se reflejan los caso de uso y su adaptación a los diferentes perfiles de usuario.

En la última sección del capítulo (*sección 4.7*) se describen las herramientas elegidas para llevar a cabo el proyecto.

4.1 Actores

En esta sección se hace una descripción de los usuarios que van a utilizar nuestra futura aplicación, definiendo sus perfiles en función del uso que le vayan a dar a la aplicación o el

tipo de acceso que usen para acceder a ella. Diferenciamos:

- **Usuario no registrado o usuario anónimo**: Podrá crear su cuenta o bien acceder a la aplicación como usuario anónimo, lo cual, solamente le permitirá consultar las noticias o eventos del tablón, pero no podrá tener interacción con el mapa, esto quiere decir que no puede ver las denuncias existentes ni añadir ninguna nueva.

- **Usuario registrado**: Una vez el usuario haya iniciado sesión, podrá acceder a las noticias, a los eventos del tablón, registrar incidencias en el mapa, ver las ya existentes y mediante un desplegable lateral podrá también consultar su información personal, las denuncias reportadas y el estado de evaluación en el que se encuentran.

- **Administrador**: Es el encargado de gestionar la aplicación, añadiendo los eventos y comunicados en cada uno de los apartados. También se ocupará de la gestión de las incidencias, descartando aquellas que no sean adecuadas y comunicando al organismo oficial aquellas que han de ser solucionadas.
  El papel del administrador no se realiza directamente desde dentro de la propia aplicación, pero es necesario recoger también su papel en el funcionamiento de esta. Su función durante el desarrollo será llevado a cabo desde la consola de Google Firebase, pero una vez terminada la implementación y despliegue de la app, se proporcionará una interfaz web de administración desde donde se podrán realizar dichas funciones.

4.2 Requisitos

Como para este proyecto se sigue la metodología Scrum, los requisitos de la aplicación son definidos y modificados a lo largo del desarrollo. Sin embargo, en este caso, algunos ya han sido definidos en el apartado de objetivos del *capítulo 1*, por lo que la mayoría, ya están en el *backlog* desde un primer momento.

Todos estos requisitos, tanto los definidos al principio como los que han surgido a lo largo del desarrollo, serán clasificados en funcionales y no funcionales.

4.2.1 Requisitos funcionales

Son todos aquellos que especifican las funcionalidades, que debe cumplir la aplicación. Se definen los siguientes:

- Debe proporcionar un sistema de comunicación directa de anuncios o comunicados oficiales de la administración local.

- Dotará a la administración de un sistema que a través de los propios usuarios pueda ser empleado como medio de obtención de información sobre los desperfectos ocasionados en la zona.
- Facilitará un listado de las noticias más recientes de la población.
- Proporcionará un tablón de eventos dentro de la región.
- Contará con un sistema que permita las siguientes características:
  - Añadir nuevas incidencias al mapa.
  - Consultar el estado de las denuncias realizadas.
  - Ver las registradas por otros usuarios.
- Compartir noticias o eventos.
- Sistema de registro y autenticación de usuarios de la aplicación.
- Registro y autenticación en la aplicación mediante servicios de terceros.
- Sistema de recuperación de contraseña mediante el correo electrónico.

4.2.2 Requisitos no funcionales

Los requisitos no funcionales se corresponden con las propiedades de la aplicación: rendimiento, seguridad, disponibilidad. No se centran en lo que hace la aplicación, sino en cómo lo hace.

- **Usabilidad**: La aplicación debe tener una interfaz limpia e intuitiva, que haga atractivo y sencillo su uso. Además debe guiar al usuario y advertir sus errores.
- **Extensibilidad**: De modo que pueda ser adaptada a cualquier entorno. Podrán integrarse las imágenes corporativas de la población donde se ubique y permitir la modificación de los colores y tipografías.
- **Modularidad**: Las funcionalidades debe ser posible activarlas o desactivarlas en función de las necesidades del cliente.
- **Seguridad**: Deben validarse los datos de usuario y la información recabada debe estar protegida.
- **Eficiencia**: La respuesta de la aplicación debe ser lo más rápida posible, incluso en condiciones desfavorables como sería la falta de internet.
- **Escalabilidad**: La herramienta debe permitir el aumento del número de usuarios de modo que permita un tráfico grande de usuarios.

4.3 Casos de uso

A continuación, se explican los casos de uso posibles para cada uno de los actores descritos en la *sección 4.1*, en referencia a los requisitos definidos en la *sección 4.2*:

4.3.1 Casos de uso para el actor Usuario no autenticado

Referidos a los usuarios que no tienen credenciales en la aplicación o que hacen uso de ella sin autenticarse.

- **Registro de usuario**: Permite dar de alta a un nuevo usuario.
  Durante la etapa de registro, se solicitan datos personales para su identificación: su nombre completo, teléfono, email y una contraseña segura para los posteriores inicios de sesión.
- **Inicio de Sesión**: Permite autenticar a los usuarios creados previamente.
  Estos acceden a través de su correo de registro y su contraseña, o bien a través de servicios de terceros.
- **Recuperación de contraseña**: Permite a un usuario ya creado previamente, recuperar su contraseña a través de su email.
- **Ver noticias**: Consultar las últimas noticias de su población.
- **Compartir noticias**: Compartir las noticias mostradas en la aplicación a través de las redes sociales.
- **Consultar tablón**: Ver los eventos que existen en el tablón.
- **Compartir elementos del tablón**: Enviar los eventos del tablón a través de las redes sociales.

En la *figura 4.1* se puede ver el diagrama que refleja estos casos de uso:

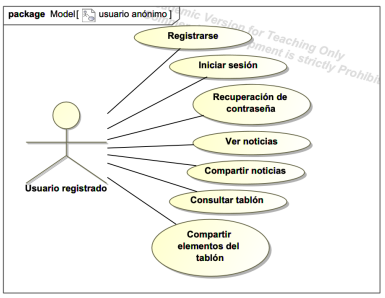


Figura 4.1: Diagrama casos de uso usuario no autenticado

4.3.2 Casos de uso para el actor Usuario autenticado

Referidos a todos aquellos usuarios que han iniciado sesión en la aplicación y por lo tanto están identificados para su utilización.

- **Ver noticias**: Consultar las últimas noticias de su población.
- **Compartir noticias**: Compartir las noticias mostradas en la aplicación a través de las redes sociales.
- **Consultar tablón**: Ver los eventos que existen en el tablón.
- **Compartir elementos del tablón**: Enviar los eventos del tablón a través de las redes sociales.
- **Ver incidencias**: Consultar las incidencias registradas por otros usuarios.
- **Añadir incidencia**: Permite denunciar las incidencias observadas. Para ello se le proporciona un formulario, donde se le pide que introduzca un título descriptivo, la dirección donde se ubica la incidencia, una breve descripción y donde puede adjuntar imágenes si lo cree necesario.
- **Consultar estado de mis incidencias**: Ver en qué estado de reparación se encuentra la denuncia (Activa, en proceso, reparada).
- **Consultar mi perfil**: Facilita conocer con qué datos está registrado en la aplicación.
- **Cerrar sesión**: Permite al usuario salir de su cuenta en la aplicación.

En la *figura 4.2* se muestra el diagrama que refleja estos casos de uso:

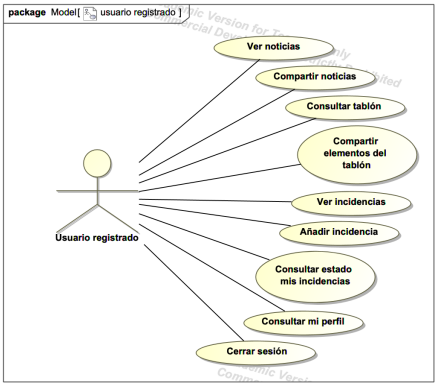


Figura 4.2: Diagrama casos de uso usuario autenticado

4.3.3 Casos de uso para el actor Administrador

Referido a los administradores de la aplicación, se definen los casos de uso que se les permite, aunque no sean implementados directamente sobre la propia aplicación.

- **Añadir evento**: Mediante un formulario el administrador puede ingresar los datos del evento, título, descripción, horario, fechas, ubicación, una imagen descriptiva y un enlace donde se pueda consultar más información.
- **Eliminar evento**: Permite suprimirlos en caso de introducción errónea o de haberse celebrado.
- **Actualizar evento**: Permite al usuario modificar la información relativa a este.
- **Consultar incidencia**: Proporciona un medio de consulta de las denuncias hechas en la aplicación, permitiendo crear informes para su solución.
  A este usuario se le proporcionan también las coordenadas exactas donde se ubica la incidencia y su informador.
- **Modificar estado incidencia**: Permite modificar su estado, para indicar que ha sido solucionada o que se está tramitando su reparación.
- **Eliminar incidencia**: Suprime aquellas que son erróneas, indebidas o que no proporcionan la suficiente información.
- **Eliminar usuario**: Eliminación de la cuenta, por motivos justificados o en caso de petición del propio usuario.

En la *figura 4.3* se muestra el diagrama que refleja estos casos de uso:

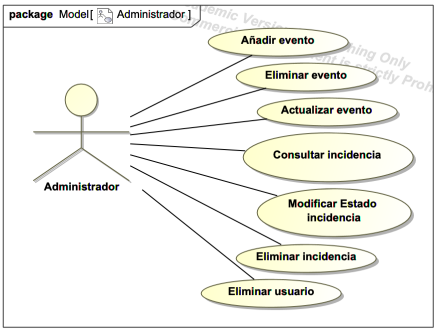


Figura 4.3: Diagrama casos de uso administrador

4.4 Flujo de la aplicación

En el diagrama de flujo se definen las secuencias para llevar a cabo todas las acciones dentro de la aplicación. En la *figura* 4.4, se ilustran las acciones, las condiciones y las transiciones de nuestra app, lo que nos permite conocer cómo se realizan y las consecuencias que estas desencadenan.

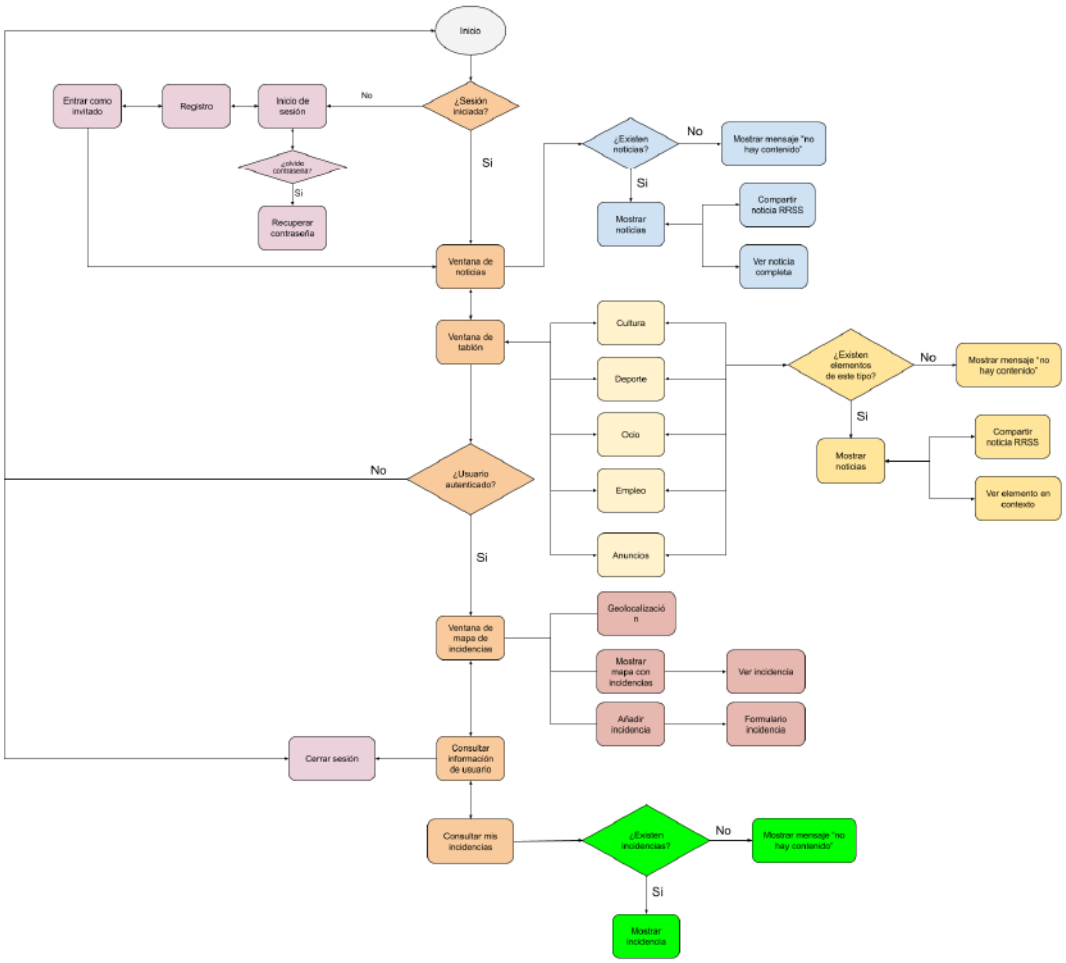


Figura 4.4: Diagrama de flujo de la aplicación

En la *figura* 4.4, se pueden ver las siguientes secuencias y comportamientos:

- La que debe seguir el usuario para iniciar sesión en la aplicación o cerrar su sesión.
- Para acceder a las noticias y el comportamiento de la interfaz.
- Para el acceso a cada uno de los elementos del tablón y comportamiento de la interfaz.
- La que se sigue para registrar una incidencia.

- Para la consulta de la información de usuario.
- Para la consulta de las incidencias propias del usuario.

4.5 Esquema de Datos de la aplicación

Todos los datos guardados, se almacenan como documentos **JSON**. La base de datos puede conceptualizarse como un árbol de documentos **JSON** alojado en la nube. Como el sistema de gestión es **NoSQL**, no hay ni tablas ni registros. Cuando se añaden elementos al árbol **JSON**, estos se convierten en nuevos nodos de la estructura y se les asocia una clave.

A continuación se puede ver el árbol **JSON** que representa la base de datos para nuestra aplicación y los campos de los documentos que lo forman:

1. **Arbol JSON:**

```
{
    "población":{
        "eventos":{
            "anuncios":{
            },
            "cultural":{
            },
            "deportes":{
            },
            "empleo":{
            },
            "ocio":{
            }
        },
        "incidencias":{
        }
        "usuarios":{
        },
    }
}
```

2. Documento JSON eventos[1]:

```
"eventos":{
    "anuncios":{ ... },
    "cultural":{
        "idEvento":<String>{
            "descripcion":"<String>",
            "fecha":"<String>",
            "horario":"<String>",
            "titulo":"<String>",
            "ubicacion":"<String>",
            "url":"<String>",
            "urlToImage":"<String>"
        }
    },
    "deportes":{ ... },
    "empleo":{ ... },
    "ocio":{ ... }
},
```

3. Documento JSON incidencias:

```
"incidencias":{
    "idIncidencia":<String>{
        "coord":"<String>",
        "descripcion":"<String>",
        "estado":"<String>",
        "fecha":"<String>",
        "imageURLS":"List<String>",
        "location":"<String>",
        "titulo":"<String>",
        "idUsuario":"<String>"
    }
}
```

4. Documento JSON usuarios:

```
"usuarios":{
    "idUsuario":<String>{
        "email":"<String>",
        "nombre":"<String>",
        "telefono":"<String>"
    }
},
```

[1]Como los documentos contenidos dentro de eventos presentan los mismos campos, solamente se muestran los de los documentos de tipo cultural para evitar redundancias

Si nos fijamos en el código de los documentos anteriores, podemos observar como para cada tipo de elemento que queremos almacenar en nuestra base de datos se crea un nodo. Por ejemplo en el caso de un evento de cultura, tendremos un idEvento, que es el identificador del nodo en el árbol y después el resto de información relativa al elemento de ese nodo: descripción, fecha, horario, título, ubicación, URL (esta es la dirección donde se podrá obtener más información) y urlToImage (es la dirección de la imagen de cabecera que queremos que se muestre).

4.6 Prototipado y diseños iniciales

Para poder realizar una aplicación con una interfaz sencilla e intuitiva, previamente es necesario realizar un diseño o prototipado de cómo será la aplicación, de modo que cuando se vaya a implementar se puedan usar como guía estos prototipos.

Estos son conocidos como **MockUps**, y pueden ser realizados a través de dibujos a papel o mediante aplicaciones específicas. En este proyecto utilizamos el editor web de *Figma*, una herramienta para el diseño de interfaces móviles o de escritorio que cuenta con elementos de diseño y estándares específicos para desarrollos con Flutter. Gracias a estas características, nos permite crear una representación de la interfaz que va a ser muy fiel al resultado final obtenido después de la implementación.

Dentro de los estándares proporcionados por *Figma*, nos centramos en la guía *Material design*, creada por Google y que es específica para crear visualizaciones en dispositivos Android. Este estándar nos marca los elementos que podemos mostrar en nuestra interfaz y la forma de hacerlo. Nos da unas pautas e indicaciones para conseguir una aplicación con un diseño limpio y sin sobrecargas, haciendo que cada elemento cumpla una función específica.

En este apartado se muestran los **MockUps** previos a la implementación de la interfaz. Siguiendo secuencias de interacción del usuario con la aplicación.
Para que los diseños sean más visuales y puedan reflejar datos reales, aunque la aplicación sea genérica y sirva para cualquier entorno, se eligió Monforte de Lemos como emplazamiento.

1. **Registro de usuario**
   En este apartado se muestran los diseño relativos al registro de usuario. En la *figura 4.5* podemos ver una primera pantalla correspondiente con la primera interacción con la aplicación, en ella se permite escoger el método de acceso, iniciar sesión si ya tiene una

cuenta, registrarse como nuevo usuario, o entrar como invitado.

En este diseño se hace el flujo de acciones para un usuario que quiere registrarse por primera vez. Una vez seleccionado el botón de *"Registrarse"*, accede a un formulario (segunda ventana de la *figura* 4.5) donde deberá ingresar sus datos para poder hacer efectivo el registro. Se comprobará que la contraseña sea lo suficientemente segura y que el email introducido sea válido y no haya sido utilizado con anterioridad. Una vez enviados los datos, el usuario accede directamente a la aplicación y se le mostrará la primera ventana, la de noticias.

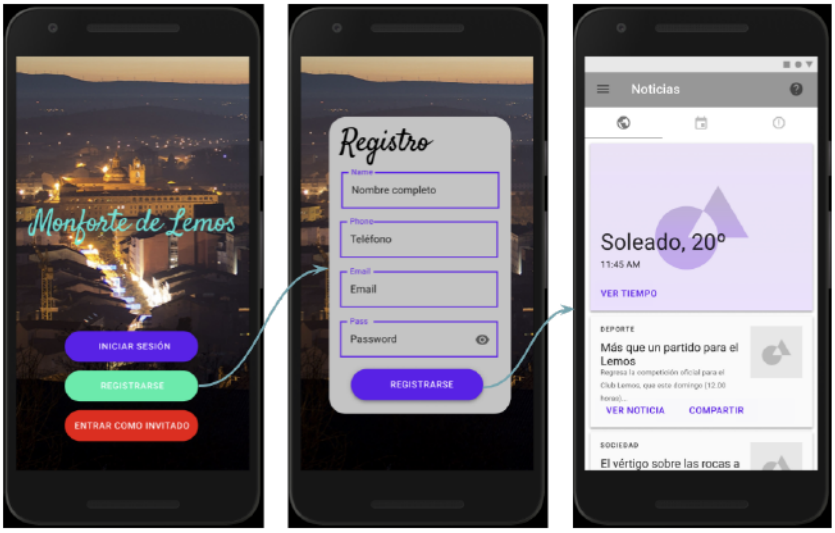


Figura 4.5: MockUps del diseño de registro de usuario

2. Inicio de sesión

En este apartado se muestran los diseños cuando ya se tiene una cuenta creada. En ellos se muestra el flujo de acciones que se debe seguir para un acceso correcto. Para ello, en la primera ventana de la *figura* 4.6, deberá pulsar el botón *"Iniciar sesión"*, una vez pulsado, accede a una nueva ventana donde se le pide mediante un formulario que ingrese su email y contraseña para poder acceder.

Otra de las opciones de inicio de sesión es mediante el botón *"Iniciar con Google"* de la segunda pantalla de la *figura* 4.6, que nos redirige al servicio de acceso de Google en el navegador web.

Una vez que se comprueban los datos e iniciamos sesión, accedemos a la primera ven-

tana de la aplicación.

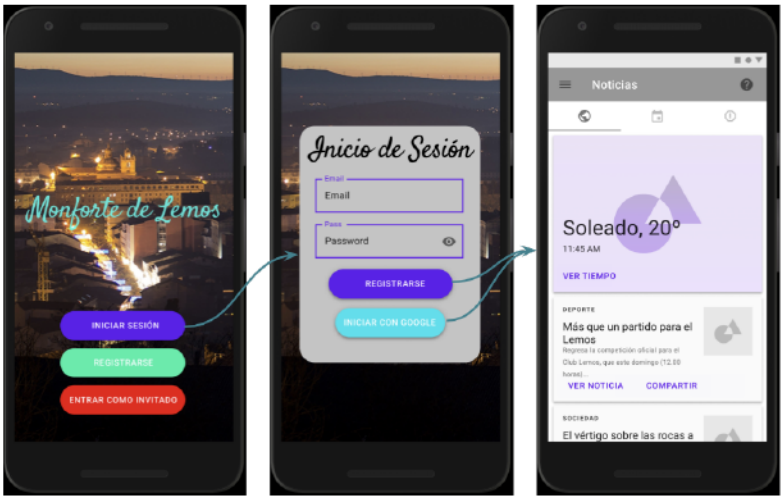


Figura 4.6: MockUps del diseño de inicio de sesión

3. **Uso de un usuario no registrado**
   Ahora se expone el diseño de los casos de uso que se pueden llevar a cabo accediendo como invitado.

   En la *figura 4.7*, se pueden ver las tres pantallas principales con las que cuenta la aplicación. Se puede acceder como invitado sin introducir ningún dato que lo identifique, pudiendo realizar las siguientes acciones:

   - Acceder a la ventana noticias, donde se le muestra un listado con las principales de la población. En cada una se proporcionan dos botones, uno para compartir a través de una aplicación de terceros su enlace y otro para acceder desde el navegador a más información.
   - Visualizar la ventana tablón de eventos, donde se puede consultar en un listado todos los actos organizados en la población. En cada uno se proporciona un botón para compartir el enlace por una aplicación de terceros. Mediante un clic sobre el evento, se accede desde el navegador a la página donde aparecen más detalles.

   Sin embargo, cuando intenta acceder a la ventana de incidencias, se despliega una alerta (tercera ventana de la *figura 4.7*), que le imposibilita el uso de esta característica y le reconduce o bien a iniciar sesión o bien a regresar a la ventana de noticias.

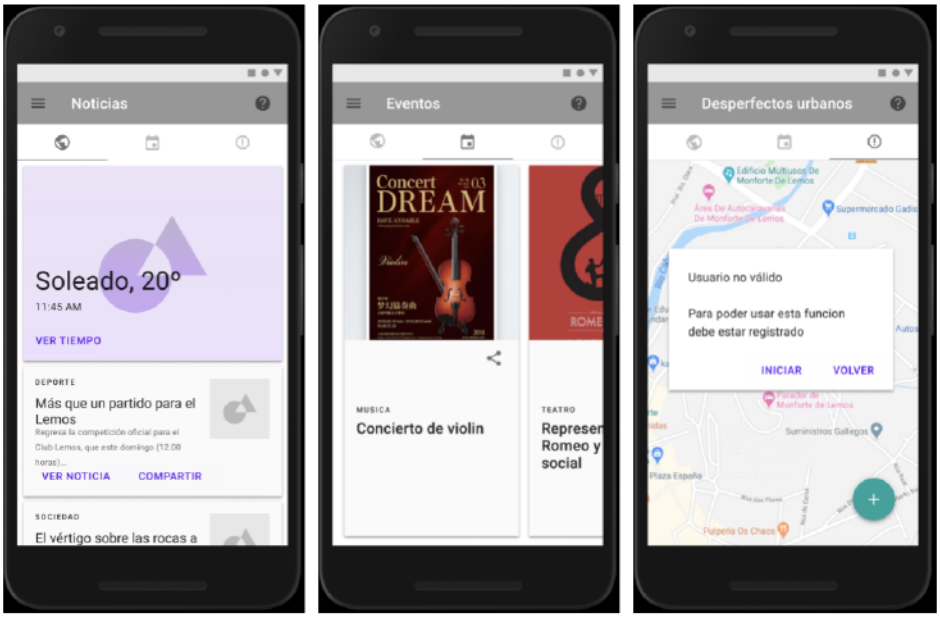


Figura 4.7: MockUps de las ventanas principales

4. **Uso de un usuario registrado**

   En este apartado, se muestran los diseños de las pantallas principales (figura 4.8) de la aplicación, para los usuarios registrados que han iniciado sesión, y los flujos de acciones que podrán realizar dentro de ellas.

   Las acciones para las ventanas de noticias y eventos son las mismas que para los usuarios no registrados:

   - En la primera imagen de la *figura 4.8* se puede ver el diseño de la ventana noticias. Donde se muestran en un listado y mediante la interacción se puede acceder a más información en la web o compartir a través de otras aplicaciones.
   - En la segunda imagen de la misma *figura*, se muestra el diseño de la ventana eventos, donde mediante un listado también, se muestran los actos organizados, cada uno podrá ser abierto en el navegador o compartido a través de una aplicación de terceros.
   - La última ventana, se corresponde con el diseño para la pantalla de incidencias, donde se muestra un mapa de la población y marcadores de otros usuarios.

   En este caso, la ventana de incidencias proporciona muchas más funcionalidades y se necesitan más diseños para las secuencias de acciones sobre ella.

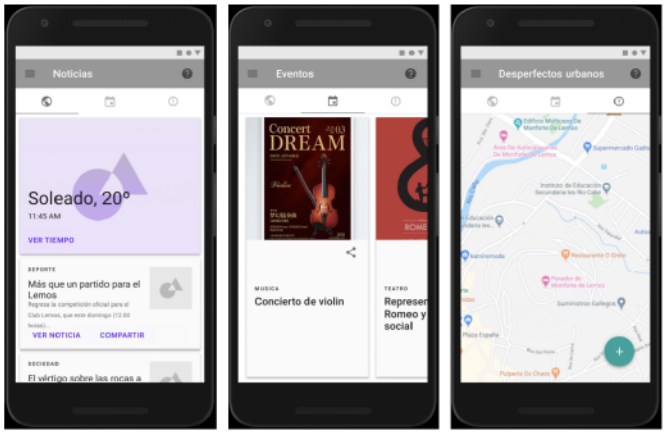


Figura 4.8: MockUps de las ventanas principales

En la *figura* 4.9, se muestra el diseño de la secuencia de acciones que debe seguir para registrar una incidencia. Primero se debe marcar en el mapa mediante un clic la posición (Marcador azul de la primera pantalla de la *figura* 4.9) y posteriormente hacer clic sobre el botón *"+"*.

Una vez pulsado, se accede a un formulario (segunda ventana de la *figura* 4.9) donde registrar los datos de la incidencia. Donde se debe indicar la calle donde se produce la incidencia y una descripción con los detalles sobre ella. Además para facilitar la descripción al usuario, se le permite incluir imágenes de la galería o tomadas directamente desde la cámara.

Una vez cubierto y enviado, el marcador cambia de color (tercera ventana *figura* 4.9) y queda registrado en la base de datos, por lo que puede ser consultado por otros usuarios.

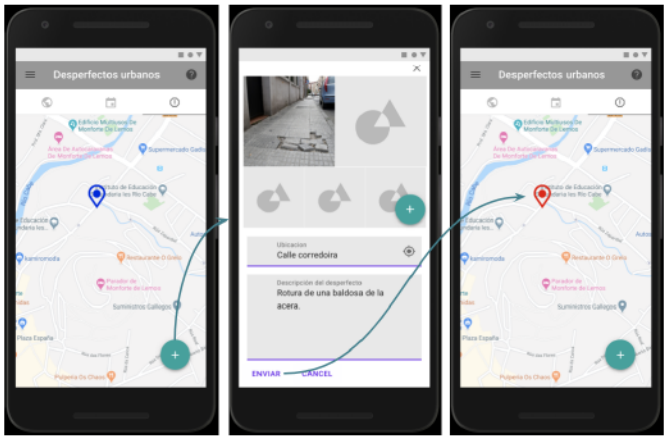


Figura 4.9: MockUps añadir una incidencia

Se pueden consultar las incidencias ya existentes para así evitar repetir denuncias. Para ello, deben hacer clic sobre los marcadores del mapa y automáticamente se les muestra una ventana superpuesta con la información de la incidencia. En la *figura 4.10*, se puede ver esta secuencia.

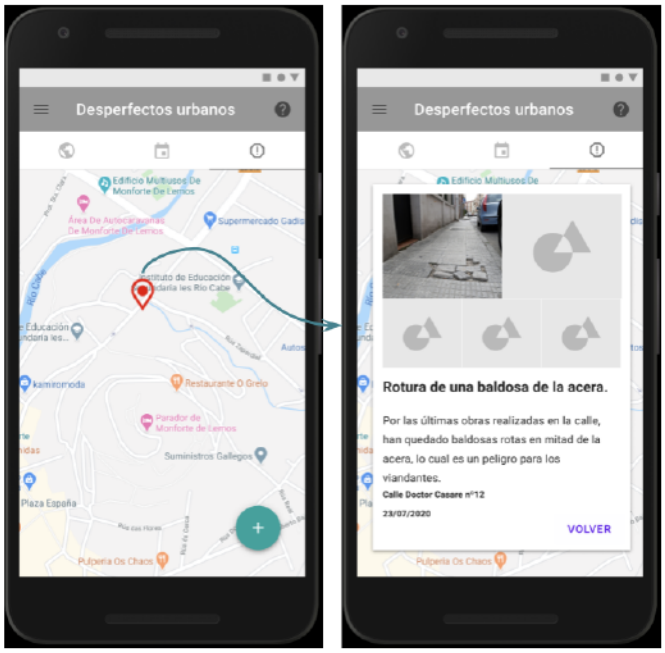


Figura 4.10: MockUps ver incidencias del mapa

Para mejorar la experiencia de uso, se proporciona un menú lateral mediante el botón de la esquina superior izquierda de la aplicación. En él se muestra información de su cuenta, unos accesos rápidos: Ajustes o preferencias, denuncias realizadas y otro a información de contacto. Desde este menú también se permite cerrar sesión mediante un botón en la parte inferior (segunda ventana de la *figura 4.11*).

En la *figura 4.11* se muestra el diseño de este menú lateral y la secuencia para consultar las denuncias registradas por el usuario. Este listado (tercera ventana de la *figura 4.11*) es privado para cada uno y ahí se muestran las incidencias y si están solucionadas o no.

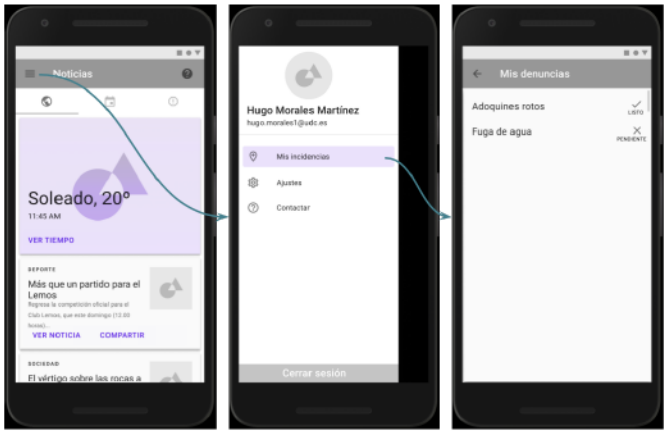


Figura 4.11: MockUps menú lateral y consulta de las incidencias del usuario

4.7 Tecnologías utilizadas

Esta sección proporciona una descripción de cada una de las tecnologías y de las herramientas empleadas para el desarrollo.

4.7.1 Flutter

Flutter [10] es un **SDK** de código abierto de desarrollo de aplicaciones móviles creado por Google e implementado en lenguaje *Dart*. Se emplea para la creación de interfaces de usuario en desarrollos a corto plazo, su principal ventaja es que con un mismo código base se pueden crear aplicaciones para múltiples plataformas Android, iOS, web y escritorio.

Flutter no emplea directamente los *widgets* nativos de iOS o Android, sino que son generados a través de su propio código. Lo que implica que los *widgets* que se empleen estén integrados dentro de la propia aplicación. Esto nos permite modificar y personalizar cualquier *widget* de un amplio catálogo o incluso crear los nuestros propios.

Flutter resulta atractivo no solamente por su portabilidad entre plataformas sino porque también aporta las siguientes características para el desarrollo:

- **Hot Reload**: al hacer algún cambio en el código se podrán ver los efectos reflejados inmediatamente, sin tener que compilar la aplicación de nuevo.
- **Acceso a las funciones nativas**: algunas funciones específicas requieren acceso a funciones nativas. Estas funciones deben implementarse mediante lenguajes nativos. Flutter permite reutilizar código *Java*, *Swift* y *Objective-C* para acceder a las funciones nativas y **SDK** en iOS y Android.

- **Facilita la creación de interfaces atractivas**: gracias a que cuenta con elementos prediseñados basados en *Material Design* o *Cupertino*, contribuye a la creación de interfaces atractivas sin un conocimiento previo de diseño.

- **Librerías de paquetes muy amplias**: al tratarse de una herramienta de código abierto, podemos encontrar infinidad de librerías desarrolladas por otros usuarios o por Google en internet, que nos ayudan en varios aspectos de la implementación, desde los elementos del diseño hasta funcionalidades de la propia aplicación. Como puede ser la traducción de **JSONs** y la integración de **APIs**.

Todas estas características hacen que el desarrollo sea mucho más sencillo y rápido que con otros **SDKs**.

4.7.2 Dart

Dart [11] es un lenguaje orientado a objetos, de código abierto desarrollado por Google para la construcción de aplicaciones web, servidores, aplicaciones de escritorio y también para hacer aplicaciones dentro del *framework* Flutter. Este lenguaje es el elegido para llevar a cabo la implementación de la interfaz de usuario y el *backend* de la aplicación, ya que dicho lenguaje es el pilar del *framework* Flutter. Dart surge como alternativa a *JavaScript* para el desarrollo de aplicaciones web, sin embargo su utilización se ha extendido más hacia la implementación de interfaces móviles.

Este lenguaje de programación nos aporta una versatilidad y velocidad de desarrollo mayor que otros lenguajes así como una curva de aprendizaje mucho menor en lo que a desarrollo móvil respecta. Dart nos permite implementar un código base capaz de ser desplegado en múltiples plataformas, Android, iOS, web y en su nueva versión beta, ya permite la creación de aplicaciones de escritorio para múltiples sistemas operativos.

Dart se compila de forma anticipada (*Ahead of time*) a un código máquina nativo (*ARM o x64*) haciendo que los tiempos de arranque de la aplicación sean mucho más bajos. Otra de las características de este lenguaje es que permite la ejecución de código de forma asíncrona, lo que mejora la paralelización de procesos dentro de la aplicación, haciendo que determinados procesos secuenciales esperen por otros y se ejecuten algunos durante esa espera, mejorando la consulta de datos y la respuesta ante operaciones que llevan un periodo largo de tiempo.

Además, Dart facilita la integración de librerías en otros lenguajes, para poder realizar funcionalidades que no sean posibles de realizar con este lenguaje.

4.7.3 Firebase

Firebase [12] es una plataforma alojada en la nube, creada por Google para el desarrollo de aplicaciones web y móviles. Esta plataforma integrada con *Google Cloud Platform* nos proporciona un conjunto de herramientas para la creación y sincronización de proyectos, que nos permitirá una mayor escalabilidad en número de usuarios en nuestras aplicaciones.
Es por esto y por otras características citadas a continuación, por lo que se ha decidido utilizar esta plataforma para alojar el *backend* de la aplicación.

Características de interés:

- **Sincronización**: Nos proporciona un medio para guardar y sincronizar datos en la nube en tiempo real.
- **Herramienta multiplataforma**: Proporciona una serie de herramientas que hacen que su integración sea sencilla y válida para múltiples plataformas, móvil, web o escritorio; y para diferentes sistemas y lenguajes.
- **Escalabilidad automática**: Gracias a las tecnologías que Google proporciona, nos permite la creación de aplicaciones pequeñas hasta aplicaciones a gran escala.
- **Análisis de datos**: Nos provee de unas métricas sobre 500 tipos de eventos.
- **Precio reducido**: Proporciona el soporte necesario para crear la aplicación de manera gratuita, creando planes acordes a las necesidades del desarrollador.
- **Monetización**: Nos provee de un recurso para ganar dinero a través de anuncios en la aplicación con *AdMob*.

Herramientas Google Cloud Platform:

Firebase proporciona una consola web, desde donde podemos acceder a nuestras aplicaciones y gestionar los servicios aplicados a cada una de ellas, de esta forma accedemos en cualquier momento y desde cualquier parte a su configuración. Los servicios que podemos gestionar desde aquí, son los siguientes:

- **Authentication**: Proporciona un servicio de gestión de usuarios multiplataforma. Gracias a esta herramienta podemos añadir a nuestra aplicación un mecanismo de autenticación y registro para poder administrar a nuestros usuarios. Permitiéndonos el registro de usuarios a través de los métodos que indiquemos: teléfono, email y contraseña, Google, Facebook, Twitter o Apple. Nos permite también añadir la necesidad de verificación

del registro para garantizar la identidad del usuario (mediante email o SMS personalizable) y de recuperación de contraseñas olvidadas.
Dentro de este servicio, también se nos provee de unas reglas definibles, para administrar los diferentes casos de registro e imponer normas de uso.

- **Cloud Firestore**: Es una base de datos, flexible y escalable, que aloja los datos en la nube en formato **NoSQL**. Sirve para mantener los datos sincronizados entre apps cliente, ya que emplea agentes de escucha en tiempo real, proporcionando los datos en tiempo real en el momento en el que se necesiten, también ofrece asistencia sin conexión para los dispositivos, lo cual dota de capacidad de respuesta incluso en condiciones de conectividad baja.
  Se utiliza para almacenar un registro de usuario y para almacenar las incidencias y eventos por localidad.

- **Realtime Database**: Es otra de las bases disponibles, de formato **NoSQL** y de almacenamiento en la nube. Esta base de datos está optimizada para ofrecer datos que es necesario actualizar, eliminar o proporcionar de forma síncrona a todos los usuarios de nuestra aplicación.

- **Cloud Storage**: Plataforma de almacenamiento en la nube proporcionada para almacenar archivos. Nos proporciona una **URL** con la que se podrá acceder a estos archivos almacenados.
  Se utiliza para almacenar las imágenes de las incidencias subidas por el usuario.

- **Cloud Functions**: Servicio encargado de alojar las funciones que se ejecutan en el *backend*. Son funciones de respuesta a determinados eventos, se pueden emplear como *triggers* de las bases de datos o para programar algunas acciones periódicas para que se realicen de forma automática. Gracias a esta funcionalidad se evitan problemas de seguridad y se agilizan los procesos automáticos.

4.7.4 NewsAPI

NewsAPI [13] es la **API** que se emplea para buscar las noticias que serán mostradas en la aplicación. Esta **API** nos proporciona a través de una petición web un **JSON**, en el que se reflejan todas las noticias que guardan relación con las palabras claves introducidas. Nos permite a través de la petición, filtrar por fecha, categoría, fuente o dominio; elegir país de la noticia, y ordenar los datos obtenidos por relevancia o siguiendo otro criterio introducido por el usuario. Este servicio es completamente gratuito durante el proceso de desarrollo y de pago posteriormente en función del uso que se le dé.

En la siguiente figura se pueden ver las principales características que ofrece esta **API**:

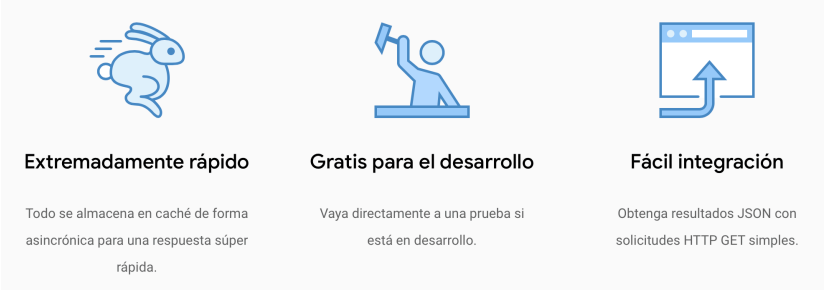


Figura 4.12: Ventajas de usar NewsAPI

Esta **API** le transfiere a nuestra aplicación una capacidad de respuesta muy alta a la hora de cargar las noticias, además, como su uso es a través de una petición **HTTP** resulta muy sencilla su integración.

4.7.5 Google Maps API

La **API** de **Google Maps** [14] es la que se emplea para poder mostrar un mapa en el apartado de incidencias de la aplicación. Mediante esta **API** podemos introducir un plano personalizado en el que el usuario final podrá marcar la ubicación donde se encuentra la incidencia. Además, gracias a las funcionalidades de Google Maps, se proporciona una herramienta de geolocalización, que permite al usuario en caso de no saber ubicar la incidencia, poder localizar su posición y desde ahí marcar la incidencia.
Esta **API** nos proporciona un crédito de **200$** mensuales, que para la etapa de desarrollo es suficiente.

4.7.6 Gestión de desarrollo y organización

En este apartado se enumeran las herramientas y aplicaciones utilizadas para la gestión y planificación de las tareas a desarrollar y también las empleadas para llevar a cabo cada una de estas tareas involucradas en la implementación.

- **Trello**: **Trello** [5] es un software de administración de proyectos que facilita la colaboración entre equipos. Gracias a la versatilidad de esta herramienta no solamente sirve para la gestión de proyectos, también se puede emplear como gestor de viajes, gestor de agenda o tareas.
  Trello permite su uso tanto en web como a través de clientes móviles; en la página principal del proyecto, nos encontramos el *board* o Tablón, donde podremos crear listas verticales con *cards*, que pueden moverse o reasignarse a otras listas. Cada una de estas *cards* se corresponde con una tarea y dentro de estas *cards* aparecen las subtareas en

las que se dividen. Estas tarjetas permiten la introducción de comentarios por lo que se podrá indicar a los otros integrantes del grupo el punto en el que está esa tarea y en que pueden ayudar.
Es empleado para gestionar la metodología de desarrollo empleada y para llevar una línea temporal del proyecto basándonos en las tareas a realizar.

- **Figma**: **Figma** [15] es un editor de gráficos vectorial web que permite el diseño de interfaces o prototipos. Esta herramienta nos proporciona un entorno de diseño gratuito. Pese a ser una aplicación web, permite simular un flujo de ejecución de la interfaz que hemos creado, pudiendo ver como se ajusta en un dispositivo que le hayamos indicado. Cuenta con librerías muy amplias de *widgets*, que ayudan a un diseño fiel a cómo sería su implementación real tanto para dispositivos Android (*Material Design*) como para dispositivos iOS (*Cupertino*). Otro de los aspectos destacables es su cariz colaborativo, que permite el trabajo en equipo de forma más eficiente.
  Esta herramienta es empleada para el diseño de la aplicación y la creación de los *MockUps*.

- **MagicDraw**: **MagicDraw** [16] es una herramienta **CASE**. Está destinada a ayudar en varios aspectos del ciclo de vida del desarrollo software, nos servirá para realizar un diseño del proyecto, mediante la documentación y creación de diagramas, previos a la implementación. Gracias a la creación de estos diagramas se podrá realizar una mejor planificación de las tareas y se podrán mejorar algunos aspectos de funcionamiento antes de empezar con la implementación, abaratando el coste humano y reduciendo también los costes económicos y temporales.

- **Android Studio**: **Android Studio** [17] es el entorno de desarrollo oficial de Android. Se emplea para el desarrollo de la aplicación, ya que cuenta con soporte para las tecnologías que usa Flutter y *plugins* para facilitar su manejo. Incorpora un emulador donde probar la aplicación mientras es desarrollada y funciones como el *Hot reload* citado anteriormente en las ventajas de Flutter. Este entorno cuenta con herramientas como inspección de elementos en la interfaz, medición de consumo de RAM del dispositivo móvil e inspección de distancias y colocación de *widgets*, que ayuda a ver errores en el diseño de la interfaz.
  Además cuenta con un control de versiones local, que nos permite revisar cambios en el código de forma local y comparar con versiones más antiguas. Desde este propio entorno se podrá administrar un control de versiones con GitLab.

- **Xcode**: **Xcode** [18] es el entorno de desarrollo propio de Apple, es de uso exclusivo en sus ordenadores (MacBook). Con él, se pueden crear aplicaciones para todos los dispositivos de la marca Apple y realizar pruebas sobre dispositivos físicos o mediante un

emulador. En nuestro caso, será empleado para probar la aplicación sobre dispositivos iOS y realizar las configuraciones necesarias para un correcto funcionamiento en dicho entorno.

- **GitLab**: **GitLab** [19] es un servicio de control de versiones y desarrollo de software basado en **Git**. Para llevar a cabo el proyecto se ha creado un repositorio en el que se ha ido añadiendo código incrementalmente cada vez que se añadía una nueva funcionalidad o se corregía algún error de funcionamiento. Aunque su principal virtud es la de facilitar el trabajo colaborativo en grupo, en este caso como el desarrollo es unipersonal, simplemente se ha utilizado para guardar versiones y poder retroceder en caso de error. Se puede utilizar desde la línea de comandos o directamente desde el entorno de desarrollo.

- **Overleaf**: **Overleaf** [20] es un editor de texto *LaTeX* de entorno web, que permite el trabajo de forma colaborativa. Se ha utilizado para la redacción de la memoria. Esta herramienta nos permite compilar el texto según lo vamos escribiendo, por lo que en todo momento estamos viendo lo que estamos escribiendo a la izquierda y su resultado a la derecha. Además gracias a su carácter web, es editable desde cualquier dispositivo y se puede acceder a él desde cualquier parte.

Capítulo 5

Diseño e implementación

Una vez definidos los requisitos de la aplicación, este capítulo describe el prototipo preliminar de la aplicación, y el diseño e implementación final de la misma. Al tratarse de un software basado en componentes (**CBSE**), ambos procesos están ligados muy fuertemente a los elementos que forman la aplicación, por lo que en este capítulo, se hará una descripción en base a algunos de estos componentes para constatar el impacto que tienen individualmente en el diseño final obtenido.

En las secciones del capítulo se tratan: el diseño general de la aplicación explicando los patrones de diseño empleados para su construcción (*sección 5.1*), la lógica de la base de datos donde se aloja la información de la aplicación (*sección 5.2*). Y finalmente, tras la explicación de las decisiones de diseño, se comentan los detalles de la implementación de algunos casos de uso y los componentes de las interfaces finales (*sección 5.3*).

5.1 Arquitectura general de la aplicación

Para poder llevar a cabo el diseño de la aplicación, es necesario definir una arquitectura a seguir. Esta proporciona información de los componentes que formarán la aplicación, la comunicación que se establecerá entre ellos y cómo funcionarán en conjunto. Siguiendo la documentación del *framework* Flutter, se recomienda emplear un patrón de diseño que desacople y separe lo máximo posible la interfaz de usuario de la lógica de la aplicación. Teniendo esto en cuenta, se recomienda la aplicación del patrón **BLoC** (Business Logic Component) o el patrón **MVVM** (Model-View-ViewModel).

BLoC se basa en la utilización de un intermediario entre las vistas y la capa modelo. Este se basa en el patrón observador, según el cual cada componente tiene un estado del que dependen el resto. Este patrón fue creado con el objetivo de diseñar aplicaciones sin una estructura

rígida como con otros patrones de diseño, permitiendo crear componentes muy grandes y con unas lógicas de negocio muy amplias, que son ágiles a la hora de recibir modificaciones. En este caso, al tratarse de una aplicación sencilla que no abarca una lógica demasiado grande, se descartó el uso de **BLoC** por simplicidad. Optándose por la utilización del patrón **MVVM**, explicado a continuación.

MVVM se basa en el patrón de diseño Model-View-Controller. Ambos son empleados en la creación de aplicaciones y se comportan de forma similar. **MVVM** presenta las siguientes diferencias y mejoras con respecto a **MVC**:

- Empleando **MVVM**, no se controlan los cambios en la vista o en los datos manualmente. Son actualizados directamente cuando sucede un cambio sobre ellos, esto quiere decir, que si se produce una variación sobre un dato de la vista, se actualiza automáticamente el modelo y viceversa. Con el patrón de diseño **MVC**, estos cambios son gestionados por el controlador y no automáticamente.
- Aunque en **MVVM** siguen existiendo tres componentes, sus funciones han variado. El modelo de vista es el único componente que accede al modelo y la vista solamente accede al modelo de vista. De este modo, la vista obtiene sus datos a través del modelo de vista y no directamente del modelo. Gracias a esta característica, la vista no tiene conocimiento de la existencia del modelo ni de las operaciones que en él se producen.
- Gracias a la característica anterior, los componentes presentan un mayor desacoplamiento que permite la utilización de diferentes tecnologías a la hora de desarrollarlos, y facilita la realización de cambios en las capas sin afectar a las demás al no existir dependencias.

Los componentes o capas del patrón **MVVM**, aplicadas a la aplicación, son los siguientes:

- **Modelo**: Se corresponde con la capa de datos con los que trabaja el sistema. El modelo contiene la información, pero no las acciones o servicios que se encargan de manipularla. Esta capa es independiente por completo de la capa de vista. La capa modelo de nuestra aplicación, es implementada utilizando Firebase y almacena todos los datos de la aplicación.
- **Vista**: Es la capa encargada de presentar la información a través de elementos visuales. Las vistas son activas, incluyen comportamientos, eventos y enlaces a datos, que de cierta forma necesitan conocer el modelo. Se ha implementado una clase por cada vista. Cada una de estas clases está compuesta por una serie de componentes, que permiten mostrar al usuario la información.

- **Modelo de vista**: Esta capa actúa como intermediaria entre el modelo y la vista. En ella reside la lógica de presentación y actúa como una abstracción de la interfaz. La comunicación con la vista se realiza a través de los enlaces de datos. En el caso de esta aplicación y para poder cumplir las condiciones de uso de este patrón, que indica que cada vista debe ir acompañada de un modelo de vista, se ha creado una clase modelo de vista por cada vista.

En la *figura 5.1*, se puede ver un esquema de la estructura que tiene la aplicación desarrollada y los elementos que compondrán cada capa según el patrón empleado.

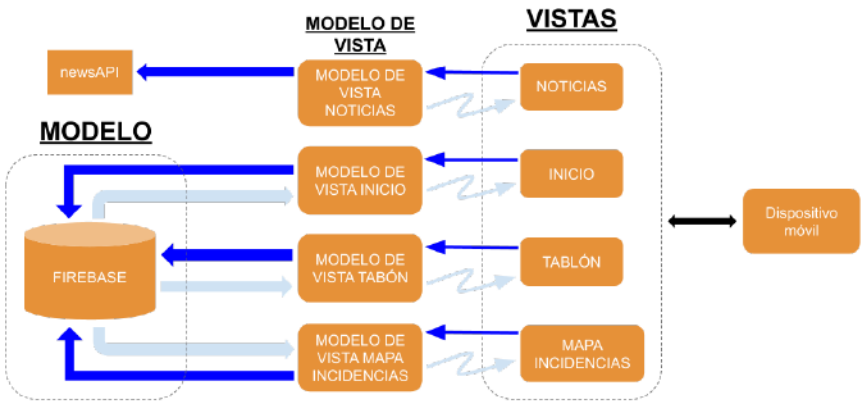


Figura 5.1: Esquema de la estructura de la aplicación

Empleando este patrón de diseño, se puede utilizar un mismo modelo de vista para varias vistas, sin embargo, de cara a ampliar la funcionalidad de cada una de las vistas en un futuro y cumplir con los requisitos del *framework* empleado, se decidió emplear un modelo de vista por cada vista. Este modelo de vista es el encargado de recuperar los datos necesarios que necesita la vista para actualizarse. El usuario interactúa con una vista, y esta se encarga de solicitar los datos a la capa modelo de vista y actualizar su estado. De esta forma, las operaciones que se realizan para la recuperación de los datos de la capa modelo son ajenas a la vista. El modelo de vista se encarga de recuperar mediante peticiones **HTTP**, a Firebase o a la **API** de noticias, la información que se mostrará en la vista. De este modo el empleo de este patrón funcionaria de la siguiente forma:

1. El usuario accede a la pantalla de eventos en la aplicación.
2. La vista se actualiza automáticamente solicitando los datos a la capa modelo de vista de eventos.
3. La capa modelo de vista solicita mediante una petición **HTTP** a la base de datos alojada en Firebase los eventos de un determinado tipo en función de la selección que haya hecho el usuario sobre la vista.

4. La capa modelo proporciona los datos solicitados a la capa modelo de vista.

5. La capa modelo de vista proporciona a la vista los datos para mostrar.

5.2 Diseño de la base de datos

La base de datos se aloja en Firebase, que como hemos explicado en el *apartado 4.7.3* es una plataforma que nos permite almacenar y gestionar bases de datos **NoSQL**. En este tipo de bases de datos, los datos no son almacenados en tablas o filas, sino que se almacenan en documentos. Estos documentos (estructurados en formato **JSON**) son organizados en colecciones. Las colecciones tienen un esquema flexible, es decir, no es necesario definir un esquema para la colección antes de insertar un documento, pudiendo tener este documento campos diferentes a los de otros documentos de la misma colección. En el *apartado 4.5* del capítulo de análisis, ya se hizo una descripción del modelo de datos generado basándonos en las especificaciones de la aplicación.

En esta sección, se realiza una explicación de las colecciones que conforman la base de datos. Aunque los campos de las colecciones puedan variar, en los siguientes apartados se hace una descripción de los campos de datos que deben almacenar los documentos en cada una de las colecciones definidas en nuestra aplicación.

A continuación en la *figura 5.2* se puede ver un esquema de estas colecciones con los documentos que las forman y los campos establecidos de forma predeterminada en nuestra aplicación.

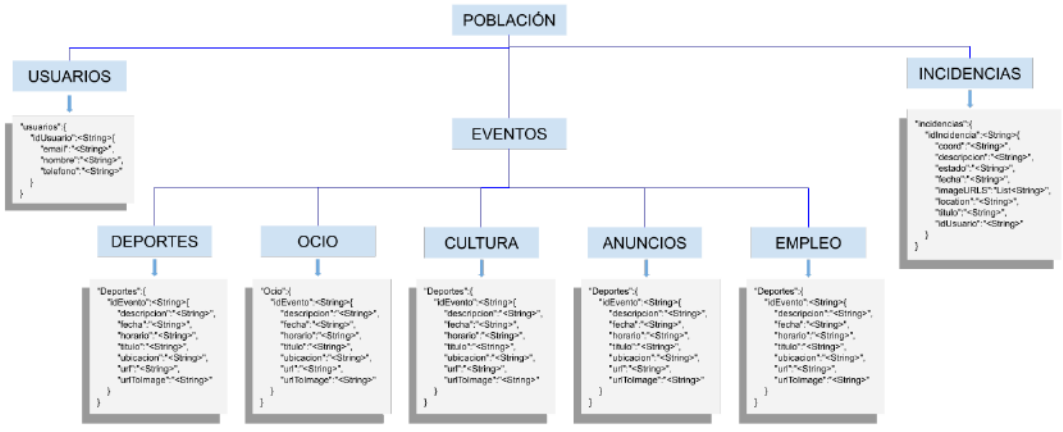


Figura 5.2: Esquema base de datos Firebase

5.2.1 Colección Usuarios

Sirve para almacenar la información de los usuarios que utilizan la aplicación. Para cada uno se genera un identificador único y se almacena: el nombre completo, un número de teléfono y un email.

Los datos de email y teléfono son empleados para la validación de la cuenta y como medio de notificación. A continuación, en la *figura 5.3* se puede ver un ejemplo de cómo muestra Firebase cada documento de la colección Usuarios dentro de su consola de administración. En la *tabla 5.1*, se hace una explicación de cada uno de los campos del documento.

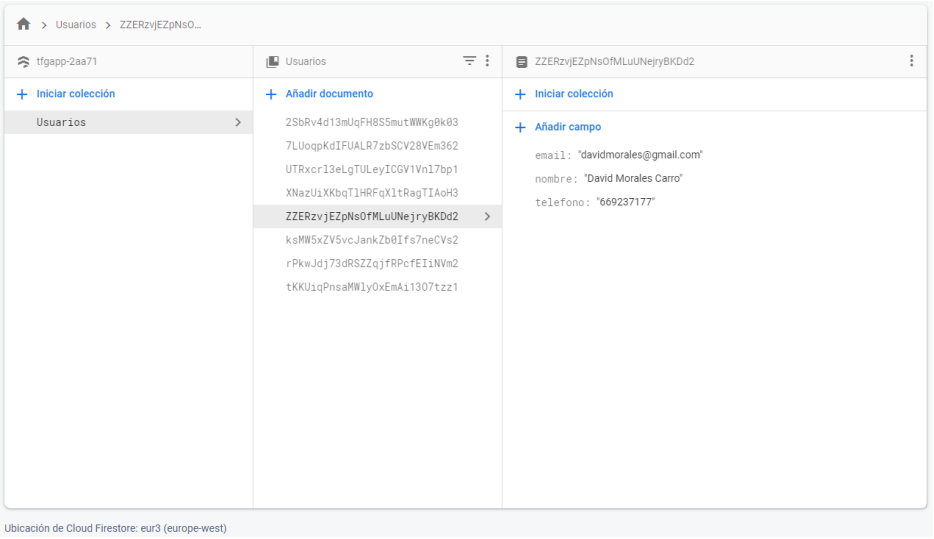


Figura 5.3: Ejemplo de documento colección Usuario Firebase

Usuarios		
Nombre	**Tipo**	**Descripción**
idUsuario	String	Clave primaria de la colección. Sirve para identificar al usuario
email	String	Email del usuario
nombre	String	Nombre completo asociado al usuario
telefono	String	Número de teléfono del usuario

Tabla 5.1: Tabla sobre la colección de datos Usuarios.

5.2.2 Colección Eventos

Se trata de una colección de colecciones, es decir, dentro de esta colección se recogen las colecciones: cultura, deportes, ocio, empleo y anuncios. Todas ellas tienen los mismos campos

y para no ser redundante en la explicación, se decidió agrupar todas bajo el mismo tipo.

Cada evento dentro de estas colecciones cuenta: Con un identificador autogenerado único, una explicación sobre sus características, las fechas en las que se va a producir, su duración, un título descriptivo, una dirección dónde se almacena la imagen descriptiva del evento, una localización de donde se va a organizar, y una dirección dónde obtener más información.

En la *figura 5.4* se puede ver un ejemplo de cómo se muestra un documento en la consola de Firebase de la colección Cultura, recogida dentro de la colección Eventos. A continuación, en la *tabla 5.2* se hace una explicación de cada uno de los campos de este documento.

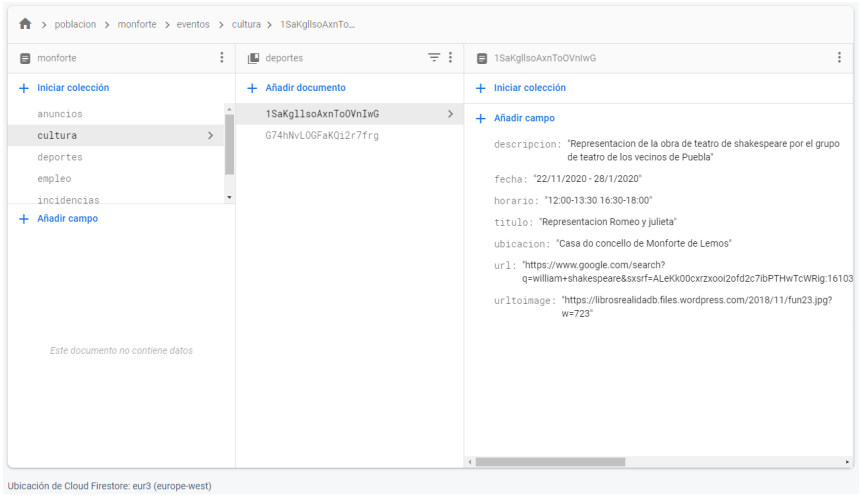


Figura 5.4: Ejemplo de documento colección Eventos Firebase

Eventos		
Nombre	**Tipo**	**Descripción**
idEvento	String	Clave primaria de la colección. Sirve para identificar los eventos
descripcion	String	Explicación sobre el evento
fecha	String	Periodo de tiempo en el que se organiza
horario	String	Duración del evento
titulo	String	Título del evento que se va a organizar
ubicacion	String	Localización del evento
url	String	URL a la página donde se explica en mayor detalle el evento
urlToImage	String	URL a la imagen de portada que queremos mostrar

Tabla 5.2: Tabla sobre la colección de datos Tablón.

5.2.3 Colección Incidencias

Se almacena la información de todas las incidencias denunciadas por los usuarios. Para cada una se genera un identificador único, y se almacena toda la información proporcionada por el usuario: un título descriptivo, una ubicación, las **URLs** de las imágenes facilitadas y una descripción sobre los motivos de la denuncia.

Además y de forma automática, la aplicación almacena: la posición del marcador en el mapa, para tener un segundo método de ubicación aparte de la dirección proporcionada en el formulario por el usuario, y una fecha para hacer más sencilla su organización.

En la *tabla 6.1* se hace una explicación de cada uno de los campos que componen el documento. A continuación, en la *figura 5.5* se puede ver un ejemplo de un documento de esta colección en la consola de Firebase.

Incidencias		
Nombre	**Tipo**	**Descripción**
idIncidencia	String	Clave primaria de la colección. Sirve para identificar las incidencias.
coord	String	Se obtiene del marcador situado por el usuario. Sirve para tener una segunda referencia y así facilitar a la administración la ubicación de la incidencia.
descripcion	String	Explicación sobre la incidencia: magnitud, urgencia o causa.
estado	String	Indica si ya ha sido solucionada o todavía está pendiente.
fecha	String	Momento en el que se denuncia la incidencia.
imageURLS	List<String>	Enlaces a las imágenes aportadas por el usuario para denunciar el desperfecto. Estos enlaces son generados por Firebase, tras almacenar las imágenes.
location	String	Dirección del incidente proporcionada por el usuario.
titulo	String	Nombre de la incidencia.
idUsuario	String	Id del usuario que realiza la denuncia.

Tabla 5.3: Tabla sobre la colección de datos Incidencias.

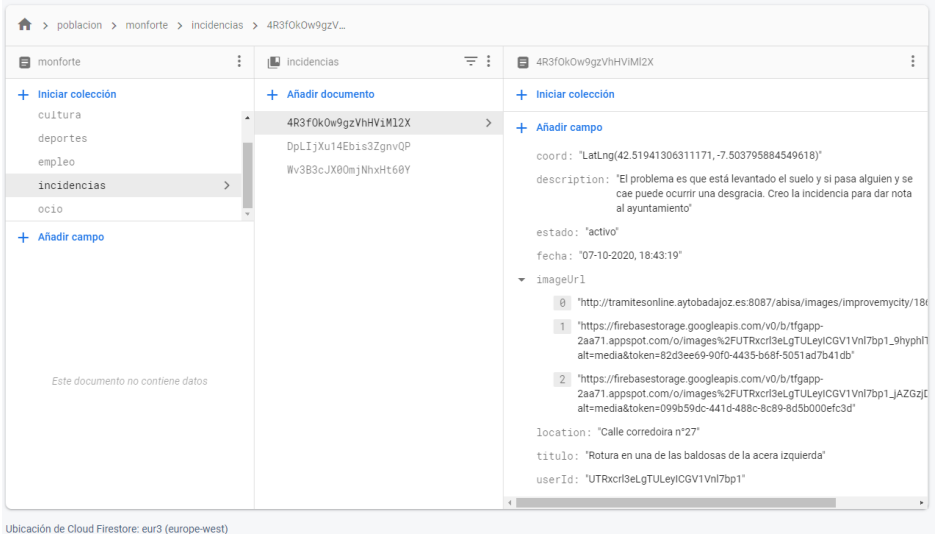


Figura 5.5: Ejemplo de documento colección Incidencias Firebase

5.3 Detalles de la implementación

En esta sección se comentan los aspectos más relevantes de la implementación de la aplicación, los problemas surgidos y cómo se abordaron.

5.3.1 Implementación de las interfaces

Como se citó en el preámbulo de este capítulo, el empleo del *framework* Flutter nos lleva a un diseño basado en componentes (**CBSE**), esto quiere decir, que para la implementación de nuestras interfaces intervienen una serie de componentes conocidos en este *framework* como *widgets*. En Flutter todos los componentes que forman parte de una vista son *widgets*, esto quiere decir, que el hecho de que la vista tenga una forma u otra depende directamente de la elección de *widgets* y de la secuencia de uso que se haga de ellos. La implementación del código de las interfaces se basa en la agregación jerárquica de componentes o *widgets* siguiendo una estructura en árboles.

En la *figura 5.6* se ve un ejemplo de cómo es la estructura de los *widgets* que componen la vista de inicio de la aplicación. La pantalla se compone de un *widget Scaffold* como base, que proporciona una ventana a la que se le puede añadir cualquier elemento. Dentro del body de este *widget* se introducen los elementos que componen la vista. En la *figura 5.6* se puede ver el árbol de *widgets* con el elemento que representa cada uno sobre la propia interfaz.

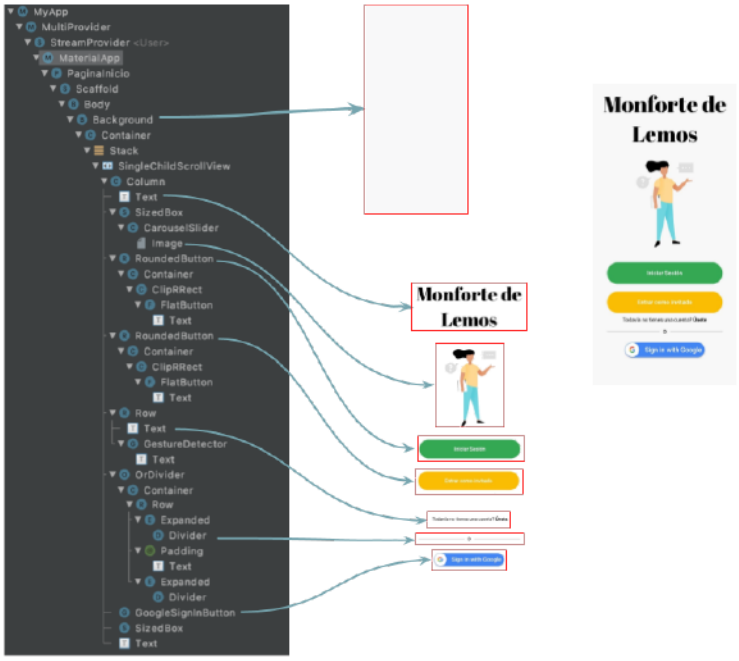


Figura 5.6: Ejemplo estructura de widgets

Cada *widget* está compuesto mediante la agregación jerárquica de otros *widgets*, lo que permite personalizar tanto como queramos cualquier aspecto de la interfaz. En la *figura 5.7*, podemos ver cómo se compone uno de los botones. Este botón es un widget creado expresamente para el proyecto y se compone de un *widget Container* con el que damos forma circular a sus bordes y cambiamos su color de fondo. En su interior contiene un *widget* necesario para centrar el botón dentro del *Container*. Para hacerlo funcional, empleamos un *widget* prediseñado (*Flatbutton*), donde podremos indicar el título del botón y hacer las conexiones para que funcione correctamente. Esta composición jerárquica de los *widgets*, nos permite aprovechar componentes básicos del *framework* para crear elementos a medida.

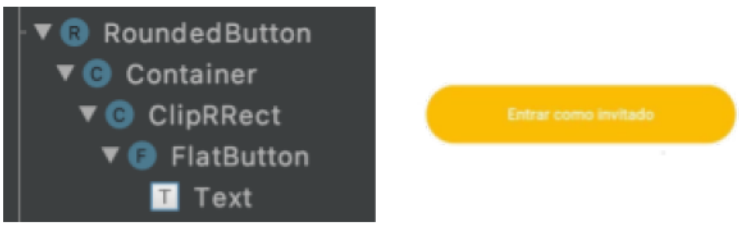


Figura 5.7: Ejemplo construcción de un widget botón

5.3.2 Implementación de los modelos de vista

Como la información está alojada en Firebase, siguiendo el patrón de diseño definido (**MVVM**), es necesario implementar unos modelos de vista que permitan controlar el comportamiento de las vistas y que controlen la información que estas deben reflejar. En este apartado se explican algunas de las implementaciones de las operaciones de la capa modelo de vista.

- Para el modelo de vista de gestión de usuarios (*Firebase Authentication*), se emplean funciones proporcionadas por la librería de *Firebase Authentication*. Con estas funciones gestionamos la creación de cuentas, el acceso de forma anónima, la recuperación de contraseñas, el inicio de sesión con email y contraseña o con la cuenta de Google y el cierre de sesión. Esta clase es la encargada de gestionar el comportamiento del usuario con la vista de inicio, la de registro y la de inicio de sesión. Encargándose de realizar la petición a la capa modelo y delimitando el comportamiento que debe tener la vista. Indicándole los datos que debe reflejar (errores o indicaciones) y en otros casos, se encarga de presentar la siguiente vista.

- Para operaciones de lectura en la base de datos de Firebase (*Cloud Firestore*), es necesario definir unas funciones específicas para obtener los datos. Por ejemplo en el caso de los eventos o de las incidencias, es necesario definir funciones que lean de la colección eventos o incidencias los datos para poder mostrarlos al usuario. En el caso de consultar las incidencias propias al usuario, es necesario definir otra función específica que filtre por el identificador de usuario. Para estos casos ha sido necesario implementar dos clases, encargadas de realizar las peticiones al modelo para recuperar la información necesaria. Una para la vista de eventos, que se encarga de hacer la petición a la colección eventos según el tipo, y devolver los eventos que deberá mostrar la vista al usuario. Y otra para la vista de incidencias, que también se encarga de hacer peticiones al modelo para extraer las incidencias que se deben mostrar en la vista. En esta clase, se define un método para recuperar la información de un usuario concreto.

- Para las operaciones de escritura sobre la base de datos *Cloud Firestore*, también es necesario crear funciones específicas, que se encargan de crear las colecciones de forma automática en caso de que estas no existan, y otras que se encargan de introducir el elemento, proporcionándole un identificador autogenerado por Firebase. Estas funciones o métodos son incluidos dentro de las clases modelo de vista. En el caso de la aplicación implementada, las únicas clases que recogen operaciones de escritura son las clases modelo de vista creadas para: las de gestión de usuario y la de incidencias.

- En el caso de proporcionar imágenes en las denuncias de incidentes. Es necesario tener

funciones que se encargan de subir dichas imágenes a *Google Cloud Storage* y generar una dirección **URL** para poder acceder a ellas. Para estas operaciones se usan las funciones proporcionadas en las librerías de *Google Cloud Storage*. Una vez almacenada la imagen, la **URL** generada se almacena dentro de la colección incidencias para posteriormente poder acceder a ellas. Estas operaciones están incluidas dentro de la clase implementada para el modelo de vista de incidencias.

5.3.3 Implementación de las noticias

Una de las funcionalidades de nuestra aplicación es el apartado de noticias. Muchos núcleos poblacionales no cuentan con una fuente de donde poder obtener estas noticias directamente y se descartó la opción de que sean introducidas manualmente por el administrador de la aplicación, por ser muy costoso y poco eficiente. Se optó por recurrir a un servicio de terceros que nos proporcione la información necesaria. *NewsAPI* nos proporciona mediante peticiones **HTTP** las noticias de la población. Esto no solamente hace que la aplicación tenga siempre las noticias actualizadas, sino que reduce posibles problemas de almacenamiento, ya que estas noticias no son almacenadas en la memoria interna del dispositivo. Estas peticiones pueden realizarse incluso en caso de tener una mala conexión a internet porque se trata de operaciones sencillas y de poca carga.

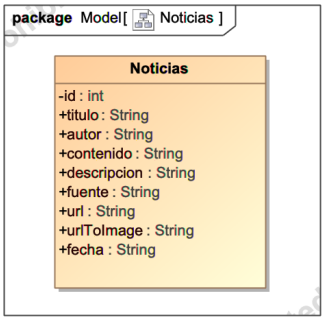


Figura 5.8: Diagrama de la clase noticias

Como el resultado de estas peticiones es en formato **JSON**, fue necesario crear funciones que extraigan la información y la transformen en objetos manejables dentro de la aplicación. Esta transformación de datos, hizo necesario crear la clase noticias, cuyas características refleja el diagrama de clases de la *figura 5.8*. Algunos de sus atributos son creados de cara a futuras mejoras en la implementación de las características de esta funcionalidad.

Esta petición se realiza de forma automática cuando el usuario accede al apartado noticias. De este modo, cuando se accede a esta sección, aparecen las más recientes. Para este proyecto, la petición es realizada desde el propio cliente móvil, pero se tiene en cuenta para su posterior

puesta en producción la utilización de *Firebase Functions*[1] para realizar la peticiones y posterior almacenamiento de resultados de forma automática desde el propio servidor. De modo, que todos los usuarios accedan a una base de datos y se reduzca el número de peticiones a la **API**. El diseño final de la interfaz que representa esta funcionalidad, difiere del diseño previo que se había hecho de ella, siendo el resultado final, el de la *figura 5.9*.

Figura 5.9: Interfaz final noticias

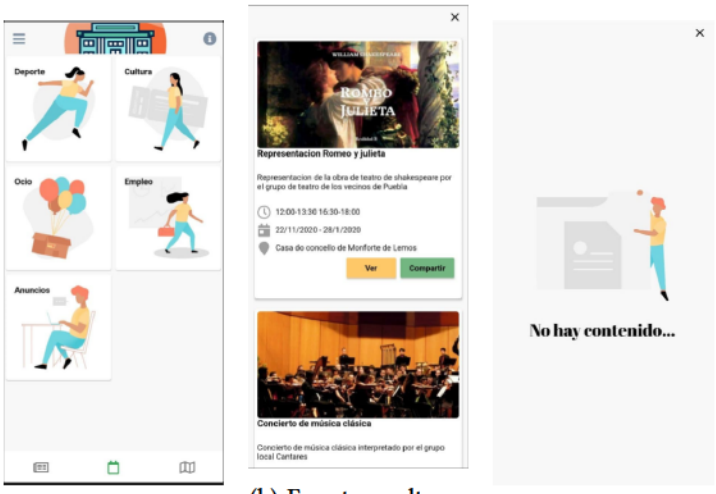
En este caso no es necesaria ninguna operación sobre la base de datos de la aplicación, porque la información de las noticias no es almacenada y tampoco la petición de permisos al usuario, puesto que tampoco se almacena ninguna información de las noticias en su dispositivo.

5.3.4 Implementación del tablón de eventos

Para llevar a cabo la funcionalidad de la ventana eventos se ha aplicado el patrón **MVVM**. Se han creado unas funciones que a partir del tipo de colección, obtienen los datos de la base de datos de Firebase. Esta obtención de datos hizo que se modificase el diseño de la interfaz, ya que resultaba más cómodo cargar los datos por categoría y no cargarlos todos y luego filtrarlos.

[1]Para este proyecto se descartó su uso al tratarse de una opción de pago.

A continuación se muestran las pantallas de la aplicación final, donde se puede observar cómo se realizan las consultas por tipo de evento. Esta modificación de la interfaz ayuda a que si en un futuro se aumenta el tipo de elementos que se quiere mostrar, no se tenga que variar toda la implementación ya hecha.



(a) Tipos de eventos (b) Eventos culturales (c) Sin eventos

Figura 5.10: Interfaces finales del apartado de tablón de eventos

En la *figura 5.10* se puede ver el resultado final y las opciones que se le mostrará al usuario en función de si existen o no eventos que mostrar. En el caso de la *figura 5.10b*, se muestran los de tipo cultural, que han sido registrados en la base de datos y en la *figura 5.12c* se muestra la pantalla que se mostraría en caso de no existir ninguno de ese tipo.

Los eventos serán introducidos manualmente por los administradores de la aplicación en Firebase, de donde se extraen y transforman en objetos manejables dentro de la aplicación. Para poder gestionar estos datos fue necesario crear la clase evento, cuyo diagrama de clases es el siguiente:

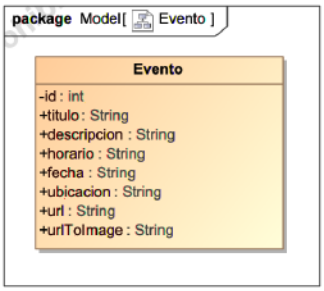


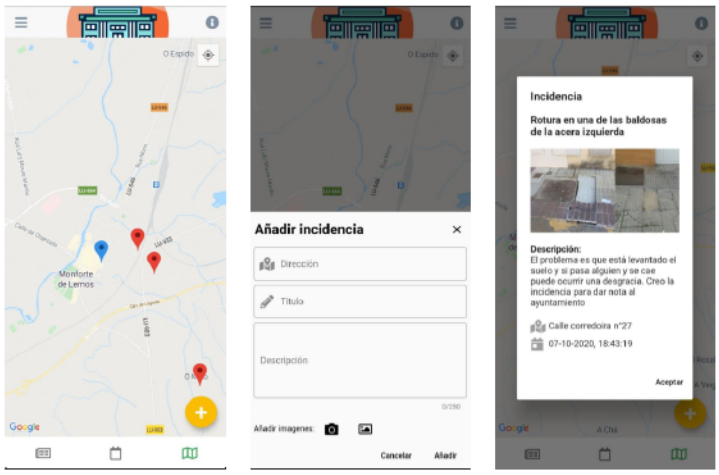
Figura 5.11: Diagrama de la clase evento

5.3.5 Implementación de las Incidencias

En la implementación de las incidencias, para poder mostrar un mapa, fue necesario integrar la **API** de *Google Maps*. Gracias a esta **API**, podemos mostrar un plano personalizado: Escogiendo entre tres tipos y pudiendo limitarlo a la población sobre la que se hace el despliegue.

El mapa desplegado por esta **API** es interactivo y nos permite solucionar uno de los problemas que se nos planteaba, el de la geolocalización del usuario. Mediante un botón en la pantalla, permitimos al usuario gracias a la ubicación **GPS** de su dispositivo obtener su posición.

La implementación de la interfaz de este apartado ha sido fiel al diseño previo que se había hecho de ella. Obteniendo el siguiente resultado final para esta interfaz:



(a) Mapa de incidencias con marcadores (b) Formulario nueva incidencia (c) Consultar incidencias existentes

Figura 5.12: Interfaces finales del apartado incidencias

En la *figura 5.12a*, se puede ver marcadas en color rojo las incidencias de otros usuarios y en azul la nuestra. Para añadir una nueva, es necesario completar el formulario de la *figura 5.12b* con los datos correspondientes. En la última figura, se puede ver un ejemplo de cómo se presenta la información de dichas denuncias.

En este caso, también ha sido necesario crear una clase específica que albergue los datos de las incidencias para poder ser enviadas a la base de datos donde serán almacenadas. En este caso, la clase creada es la siguiente:

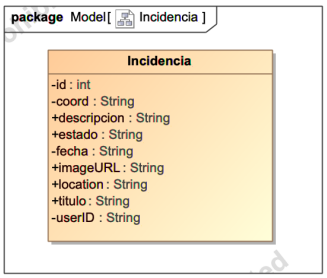


Figura 5.13: Diagrama de la clase incidencia

Para este apartado fue necesario añadir la solicitud de permisos de acceso a la ubicación **GPS** del dispositivo y solicitar los permisos de acceso al almacenamiento interno y a la cámara, para obtener las fotos en el formulario de incidencias (*figura 5.12b*).

Capítulo 6

Pruebas

En este capítulo de la memoria, se hace una explicación sobre las distintas pruebas que se han hecho sobre la aplicación. Mediante estas pruebas se intenta comprobar el cumplimiento de los requisitos de la herramienta y su correcto funcionamiento.

Este capítulo se divide en dos secciones, la primera sección (*sección 6.1*) se corresponde con las pruebas funcionales y la segunda con las pruebas no funcionales (*sección 6.2*). Dentro de cada una de estas secciones se explica en mayor detalle las pruebas concretas que se han realizado.

6.1 Pruebas funcionales

Son aquellas que se basan en la ejecución, la revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Existen diferentes tipos de pruebas, pero para este proyecto, se han realizado las que a continuación se describen en las siguientes secciones:

6.1.1 Pruebas unitarias

Las pruebas unitarias son aquellas que se encargan de comprobar el correcto funcionamiento de una unidad concreta de código. En este proyecto, se ha aplicado un test unitario sobre cada función de la aplicación y sobre cada *widget* implementado.

En Flutter contamos con herramientas específicas para estas pruebas. Un paquete test que nos proporciona el marco principal para desarrollar las pruebas unitarias y la biblioteca *flutter_test* que proporciona las utilidades adicionales para probar los *widgets*. Gracias a estos dos paquetes se han podido escribir una serie de pruebas que comprueban que las clases, funciones y métodos funcionan correctamente. Además, gracias a los métodos específicos de la clase *WidgetTester* de la biblioteca *flutter_test*, podemos comprobar la respuesta de los *widgets* a las

interacciones del usuario, pudiendo ver como responde cada uno a: introducción de texto, pulsación y arrastre.

Dentro de las pruebas realizadas sobre los *widgets*, son representativas las pruebas realizadas sobre los *widgets* de introducción de texto de las pantallas de inicio de sesión y registro. En estos test se ha simulado la entrada errónea en algunos de los campos, para comprobar que se tienen en cuenta todas las excepciones.

A continuación se puede ver un ejemplo de implementación de uno de estos métodos. Este en concreto, se utiliza para comprobar la introducción incorrecta de un email en la pantalla de registro:

```
testWidgets('Invalid email', (WidgetTester tester) async {
  var textField = find.byValueKey('emailRoundedButtonKey');
  await tester.tap(textField);
  await tester.enterText('ValorNoValidoDeEmail');
  await tester.waitFor(find.text('No es posible registrarse'));
});
```

Conclusiones de las pruebas:
Tras realizar las pruebas unitarias sobre la aplicación, se encontraron varios fallos, dos de ellos fueron:

- Uno de los botones de la pantalla de registro no funcionaba correctamente. En el formulario de ingreso de los datos de usuario, cuando se introducía la contraseña y se pulsaba en el botón para hacerla visible, si se repetía esta acción más de una vez, dejaba de funcionar correctamente.
- Al cerrar sesión después de entrar con la opción de entrar como invitado, no se regresaba a la ventana de inicio de la aplicación.

6.1.2 Pruebas de integración

Las pruebas de integración se realizan una vez se ha finalizado con las unitarias. El objetivo de estas es comprobar el correcto ensamblaje entre los distintos componentes, con el fin de comprobar que interactúan correctamente en conjunto.

Los test de integración funcionan de la siguiente manera: primero, hacen un despliegue de la aplicación creada sobre un dispositivo o emulador real y posteriormente desde una *suit de pruebas* se guía a la aplicación sobre diferentes pruebas individuales para verificar que todo

esté correcto en los diferentes flujos de la aplicación. El proceso de despliegue es independiente del proceso encargado de guiar las pruebas sobre la aplicación.

Para la realización de estas pruebas se ha utilizado el paquete *flutter_driver*. Que nos proporciona unas herramientas para crear aplicaciones instrumentadas y una *suit de tests* para poder probarlas.

Conclusiones de las pruebas:

Con la última versión implementada, no se obtuvo ningún resultado negativo que se pueda reflejar en estas pruebas.

6.1.3 Pruebas de regresión

Estas pruebas se realizan durante el desarrollo del proyecto y sirven para garantizar el correcto funcionamiento de la aplicación después de realizar algún cambio, evitando así que se produzcan errores que se consideraban ya resueltos o que no existían anteriormente. Durante el proceso de desarrollo se han realizado de forma periódica pruebas de integración sobre todos los componentes.

Una vez realizado el desarrollo y obtenido una aplicación funcional. Encuadradas dentro de este tipo de tests, se han realizado una serie de pruebas manuales, destinadas a comprobar flujos completos de interacción con la aplicación. Elaborando el siguiente plan de pruebas:

Plan de pruebas de regresión			
ID	**Descripción**	**Resultado esperado**	**Resultado**
01	Creación de una cuenta	El usuario crea una cuenta con éxito	Correcto
02	Recuperar la contraseña	El usuario recibe un mail y puede cambiar su contraseña	Correcto
03	Iniciar sesión con mail y contraseña	El usuario inicia sesión con éxito	Correcto
04	Iniciar sesión con Google	El usuario inicia sesión con éxito	Correcto
05	Ver listado noticias	El usuario puede ver un listado de noticias de su población	Correcto
06	Ver noticia completa	El usuario puede acceder a la noticia completa desde su navegador	Correcto
07	Compartir noticia	El usuario puede compartir la **URL** de la noticia a través de otra app	Correcto

08	Ver categorías del tablón	El usuario ve una cuadrícula de diferentes categorías de eventos	Correcto
09	Ver eventos por categoría	El usuario puede acceder a un listado de eventos de cada tipo	Correcto
10	Ver evento	El usuario puede acceder a toda la información del evento desde su navegador	Correcto
11	Compartir evento	El usuario puede compartir la **URL** del evento a través de otra app	Correcto
12	Ver y navegar en el mapa incidencias	El usuario puede ver un mapa de su comarca y puede desplazarse dentro de los límites de su población	Correcto
13	Colocar marcador	El usuario puede colocar un marcador sobre el mapa	Correcto
14	Acceso formulario de incidencia	El usuario puede acceder a un formulario donde completar la información de la incidencia que quiere registrar	Correcto
15	Subir imagen de galería	El usuario puede cargar una imagen desde la galería al formulario de incidencia	Correcto
16	Subir imagen desde cámara	El usuario puede adjuntar una imagen directamente de la cámara al formulario de incidencias	Correcto
17	Cargar una incidencia	El usuario crea una incidencia y se carga de forma exitosa	Correcto
18	Ver incidencias del mapa	El usuario puede consultar la información de las incidencias registradas sobre el mapa	Correcto
19	Consultar estado mis incidencias	El usuario puede consultar el estado de las incidencias que ha denunciado	Correcto
20	Consultar información personal	El usuario puede consultar los datos con los que se ha registrado	Correcto

Tabla 6.1: Tabla sobre la colección de datos Incidencias.

Conclusiones de las pruebas:

Gracias a estas pruebas se ha evitado encadenar errores, ya que con algunas nuevas funcionalidades añadidas se alteraba el funcionamiento de algunas funciones ya calificadas como correctas.

Y se han podido solucionar algunos errores como por ejemplo, el error que se producía en el inicio de sesión de usuario: En este inicio, se realiza una petición a la base de datos para obtener la información de usuario, pero en la funcionalidad de consultar los datos de usuario, se repetía dicha operación de recuperación de datos, siendo innecesaria.

6.1.4 Pruebas de aceptación

Estas pruebas tiene como objetivo validar que la aplicación cumple con los requisitos especificados por el cliente y que este determine su aceptación. En este tipo de pruebas, la ejecución y aprobación final corresponde al cliente, que es el que válida y verifica que se cumplen con sus necesidades.

Conclusiones de las pruebas:
En el caso de este proyecto, se ha instalado sobre un dispositivo final una versión con todas las funcionalidades y se ha presentado ante el tutor del proyecto. A pesar de que los MockUps presentados durante el análisis difieren del diseño final de la aplicación, se cumple con todos los casos de uso y por tanto las pruebas de aceptación son favorables. El product owner da el visto bueno a la aplicación y se considera que la prueba de aceptación está superada.

6.2 Pruebas no funcionales

Las pruebas no funcionales son aquellas que se encargan de verificar que los requisitos no funcionales son cumplidos correctamente, como por ejemplo la disponibilidad, accesibilidad, usabilidad, mantenibilidad, seguridad y rendimiento.

En el caso de este proyecto, se han aplicado las siguientes pruebas para comprobar y calificar el comportamiento de la aplicación:

6.2.1 Pruebas de usabilidad

Las pruebas de usabilidad consisten en observar el uso de la aplicación que hace un grupo de usuarios. A este grupo se les pide que realicen tareas para las cuales fue diseñada la aplicación. Mientras, se toma nota de las interacciones que estos usuarios hacen, registrando los errores y dificultades que se encuentren.

Conclusiones de las pruebas:
Para realizar esta prueba, se ha instalado la aplicación en los dispositivos móviles de un grupo de usuarios y se les ha pedido que hagan determinadas tareas. En este caso se ha seguido el

mismo plan de pruebas que el de la *tabla 6.1*. Pidiendo a cada usuario que ejecutase cada una de las tareas de dicha tabla.

Gracias a su opinión y experiencia de uso, se han modificado algunos aspectos de la interfaz para hacerla más intuitiva y bonita. Para poder recoger la experiencia de interacción de los usuarios con la aplicación, se les ha realizado una encuesta y se ha registrado los resultados en la siguiente tabla (*tabla 6.2*).

Opiniones de los usuarios			
ID	**Cuestión Propuesta**	**Respuesta afirmativa**	**Observaciones**
01	¿Considera que la aplicación tiene un diseño atractivo?	3/5	Algunos colores hacen que la aplicación no sea clara
02	¿Conoce o ha empleado alguna aplicación similar alguna vez?	3/5	Usuarios de la aplicación **Avisos Madrid** [4]
03	¿Considera que la aplicación es intuitiva y fácil de utilizar?	5/5	Funcionalidades marcadas y sencillas de manejar
04	¿Le ha resultado sencillo el proceso de registro/autenticación?	5/5	Ágil, rápido e intuitivo
05	¿Considera que la aplicación responde de forma ágil?	5/5	Ninguna
06	¿En los mensajes de error, considera que se aporta la suficiente información?	4/5	Mejorar especificación de errores en el registro de incidencias
07	¿Se ha adaptado la aplicación a su dispositivo de forma correcta?	5/5	Ninguna
08	¿Funcionan todas las características probadas correctamente?	5/5	Ninguna
09	¿Le resultan útiles las herramientas proporcionadas por la aplicación?	4/5	Necesario incluir alguna función más
10	¿Recibió alguna respuesta de la aplicación inadecuada o errónea?	4/5	Problema con el inicio de sesión de Google devuelve error no controlado

Tabla 6.2: Tabla de opinión de los usuarios involucrados.

Gracias a estas pruebas también se ha podido comprobar el comportamiento de la aplicación sobre varios dispositivos. Esta experiencia no puede considerarse como pruebas de compatibilidad. Pero han proporcionado datos interesantes sobre el comportamiento de la interfaz a tener en cuenta. A continuación, en la *tabla 6.3* se pueden ver los modelos empleados con sus resoluciones y las observaciones obtenidas.

Dispositivos empleados			
ID	**Modelo dispositivo**	**Resolución (Píxeles)**	**Observaciones**
01	Xiaomi MI9	2.340 × 1.080	-
02	Xiaomi Mi A2	2.160 × 1.080	Necesario retocar las tarjetas de noticias.
03	Samsung galaxy S8+	2.960 × 1.440	-
04	Samsung galaxy Note 10+	3.040 × 1.440	Necesario retocar la cuadrícula del tablón.
05	iPhone 11	1.792 × 828	-

Tabla 6.3: Tabla de dispositivos empleados.

6.2.2 Pruebas de escalabilidad

Las pruebas de escalabilidad son aquellas que miden la capacidad de respuesta de la aplicación ante el incremento del número de usuarios que hacen peticiones sobre ella. Esta prueba no ha sido posible realizarla a gran escala, pero se ha comprobado con múltiples usuarios al mismo tiempo. Gracias al uso de Firebase como base de datos, según su documentación, tenemos soporte para poder realizar hasta un millón de conexiones simultáneas.

6.2.3 Pruebas de rendimiento

Las pruebas de rendimiento son las pruebas que se realizan para medir la velocidad de respuesta que tiene la aplicación para una determinada tarea en unas condiciones particulares de trabajo.

Conclusiones de las pruebas:
En nuestro caso, mediremos la velocidad de todos los casos de uso de la aplicación, condicionando su medición al tipo de conexión del dispositivo. Se ha medido con conexión de datos móviles y con conexión WIFI. Obteniéndose los siguientes resultados:

Casos	**Conexión móvil** (segundos)	**Conexión WIFI** (segundos)
Iniciar aplicación	2	2
Registro de usuario	4	3
Inicio de sesión con email y contraseña	1	1
Inicio de sesión con Google	6	5
Acceder como invitado	1	1
Carga de noticias	2	2

Casos	**Conexión móvil** (segundos)	**Conexión WIFI** (segundos)
Carga de eventos	2	2
Carga de mapa	3	3
Carga de mis incidencias	2	2
Carga de información de usuario	2	2
Cierre de sesión	1	1

Tabla 6.4: Tabla de rendimiento de la aplicación.

Los datos obtenidos, reflejados en la *tabla 6.4*, demuestran que la velocidad de respuesta es óptima y por lo tanto se considera la prueba como superada satisfactoriamente.

Nos ha servido también para medir el consumo de internet que implica el uso de la aplicación en ambos casos. Siendo de **1.6 MB** el gasto tras haber ejecutado todos los casos reflejados en la *tabla 6.4*.

Capítulo 7

Conclusiones

En este último capítulo de la memoria, se hace una valoración del trabajo realizado durante el desarrollo y se exponen los siguientes pasos a seguir para mejorar la aplicación con el objetivo de su puesta en producción.

En la *sección 7.1* se describen las conclusiones obtenidas durante la realización del proyecto y se hace una valoración del resultado final obtenido. En la última sección de este capítulo (*sección 7.2*), se marcan unas líneas futuras de trabajo con el objetivo de mejorar las funcionalidades que ofrece la aplicación y de cara a poder realizar su despliegue.

7.1 Conclusiones y valoración del proyecto

De las conclusiones extraídas de este proyecto, quizás destaque el hecho de haber logrado una aplicación que implica cierta complejidad en un breve periodo de tiempo. Aunque el resultado final obtenido cumple con todos los objetivos marcados, todavía faltan muchos detalles por pulir antes de poder ser comercializado. Se considera también que los requisitos funcionales y no funcionales están cubiertos, pero se cree necesario ampliarlos para ofrecer un mejor funcionamiento de la aplicación y un abanico más amplio de funcionalidades.

La duración del proyecto se ha visto modificada en cuanto a los plazos de entrega, teniendo que retrasarlo una convocatoria. Esto me ha permitido perfeccionarlo, corrigiendo fallos y revisando detalles de diseño y contenido que por necesidad de adaptarlo a un plazo, no terminaban de resultar óptimos dentro de la aplicación. Cabe mencionar que se ha seguido una planificación diaria de las tareas para poder mantener un orden y una rutina, centrando cada día en unos puntos concretos.

La experiencia de uso del framework Flutter ha sido realmente positiva. Aunque su versión

estable es relativamente nueva, existe una amplia documentación proporcionada por Google, que hace su utilización mucho más sencilla, ayudando a conseguir un aprendizaje mucho más fluido y que sirve de guía para los desarrollos. Además, existe una comunidad de usuarios cada día más grande, que aporta infinidad de paquetes de software libre, que pueden ser integrados o que pueden servir de base para algunos elementos de nuestra aplicación. Estas características han hecho posible un aprendizaje rápido y facilitaron la prevención de puntos muertos en el desarrollo.

El objetivo principal era desarrollar una solución que fuese de fácil acceso y uso para cualquier usuario, independientemente de su edad o conocimientos tecnológicos. De esta forma se consigue que todos los individuos que forman la comunidad se sientan miembros de ella, sin dejar a nadie al margen, y que estos puedan estar al día de todas las novedades y sientan que su opinión cuenta, manifestando sus deseos de mejorar su entorno mediante la denuncia de desperfectos que les afecten en su día a día.

7.2 Líneas futuras

Como antes se ha mencionado, a la aplicación aún le queda recorrido para convertirse en un producto final. Entre esos puntos de mejora, destacan los siguientes:

Mejorar la petición de noticias:
Para evitar que aparezcan noticias no adecuadas o que no se correspondan con la población. En algunos caso, puede ocurrir que existan varias poblaciones con el mismo nombre. Por lo tanto, es necesario redefinir la petición a la **API** de *NewsAPI*, para que en cada población se muestren las noticias de forma correcta.

Otra de las opciones estudiada, es integrar la posibilidad de obtener las noticias directamente de una fuente de información de la propia población, sin tener que recurrir a una **APIs** de terceros.

Mejorar las opciones del tablón:
En este apartado surgen varias ideas de opciones que añadir a las funcionalidades existentes en el apartado tablón de la aplicación. Estas nuevas funcionalidades ayudan a hacerla más atractiva y a poder abarcar más necesidades de la población. Por falta de tiempo y por su complejidad no ha sido posible incorporar estas mejoras a la aplicación. Algunas de estas mejoras y funcionalidades son las siguientes:

- Apartado que permita realizar encuestas. Esta funcionalidad ayudaría a integrar de una forma más directa a los habitantes de la población. Permitiendo a la administración de esa población sacar a votación temas donde se requiera la participación de los vecinos.
- Apartado desde donde los vecinos puedan realizar trámites en las instituciones locales. Proporcionar una herramienta desde donde poder realizar directamente solicitudes administrativas: permisos municipales, solicitud de ayudas, padrón, etc.
- Apartado de información de transporte público. En el caso de algunas poblaciones, es interesante incorporar un apartado desde donde se pueda consultar horarios y paradas del transporte público.
- Apartado de servicios de guardia. En este apartado se tiene como idea principal, ofrecer información sobre servicios de primera necesidad o de emergencia: farmacia de guardia, servicios sanitarios, etc.
- Apartado de objetos perdidos. Para permitir a los vecinos de la población: denunciar el extravío de objetos, publicar anuncios con objetos encontrados y ver en un listado los objetos encontrados.
- Apartado de puntos de interés en la comarca. Esta opción fue pensada para ampliar el uso de la aplicación a los turistas que visiten la población. En este apartado se podrá consultar la información de los puntos turísticos: una descripción del punto, horario de apertura y cierre, localización y solicitud de entradas si fuese necesario.
- Apartado de comercio de productos locales. Esta opción también fue pensada de cara a ampliar el uso de la aplicación al turismo. Desde este apartado, se listarán productos propios de la artesanía de la zona y donde poder adquirirlos.
- Apartado de información necrológica. Ofreciendo un servicio actualizado diariamente de los decesos de la comarca.
- Apartado de reservas de instalaciones municipales. Con esta opción se proporcionaría a los vecinos un sistema desde donde poder comprobar la disponibilidad de las instalaciones municipales (pista de atletismo, pista de tenis, polideportivo, piscina, etc.) y desde donde poder realizar reserva de ellas.

Mejorar la información del mapa de incidencias:

En el caso del mapa de incidencias, una de las mejoras a realizar es la categorización de los desperfectos. De este modo, su gestión se realizaría de una forma más ágil y se le podría presentar al usuario un formulario más especifico para el tipo de incidencia que indique. Esta

categorización permite agilizar la resolución de las incidencias marcadas como urgentes.

Otro de los aspectos a mejorar en el mapa es la creación de marcadores propios para cada categoría. Por lo tanto, según el icono, se podrá distinguir el tipo de incidencia.

Herramienta web de gestión de la app:
De cara a la comercialización y puesta en producción de la aplicación, es necesario crear un software de gestión, para poder administrar la aplicación de forma más sencilla y atractiva. En el *apéndice A* se incluye un diseño de las ventanas que deberá tener dicha herramienta.

Este software puede ser perfectamente suplido mediante una aplicación web. Desde ella, el usuario administrador podrá manejar de forma más intuitiva las bases de datos, obtener datos relacionados con el uso de la aplicación, gestionar las incidencias y seleccionar las funcionalidades a mostrar en el apartado tablón de la aplicación.

Apéndices

Apéndice A

Diseños adicionales plataforma web

En este apéndice, se incluyen los prototipos de la plataforma web destinada a la administración de la aplicación. De cara a la puesta en producción de la aplicación es necesario la creación de una plataforma con la que poder hacer las tareas de administración. Es por eso, que aunque no se incluya en el cuerpo de la memoria, es importante incluirlo en un apéndice de cara a una mejor comprensión del alcance del proyecto.

A.1 Inicio de sesión

Figura A.1: Prototipo inicio de sesión

A.2 Administración apartado Noticias

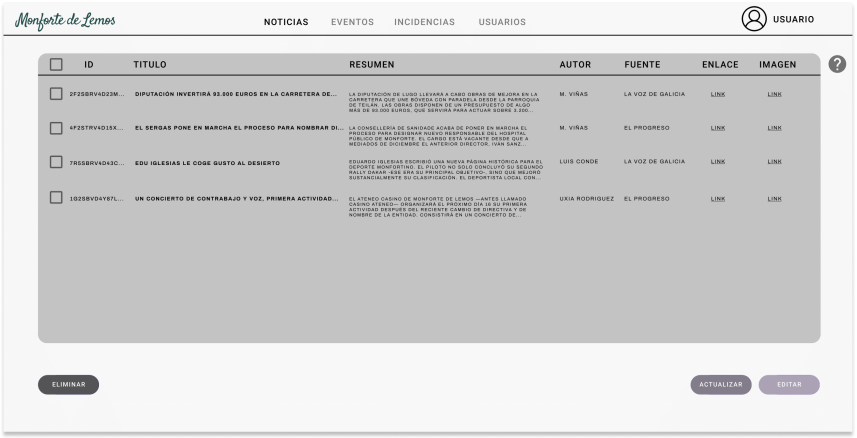


Figura A.2: Prototipo parrilla noticias

Figura A.3: Prototipo edición de noticias

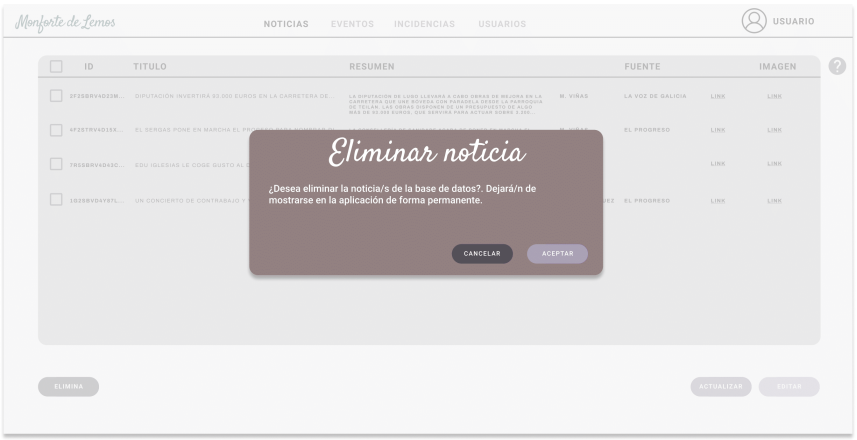


Figura A.4: Prototipo eliminación de noticias

A.3 Administración apartado Eventos

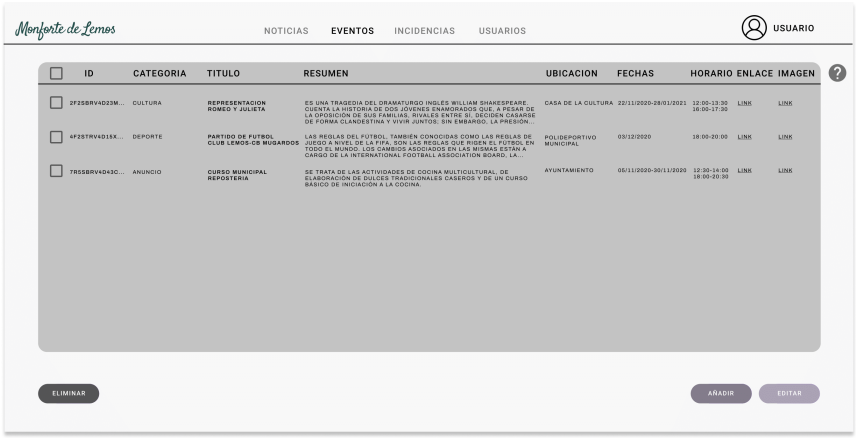


Figura A.5: Prototipo parrilla eventos

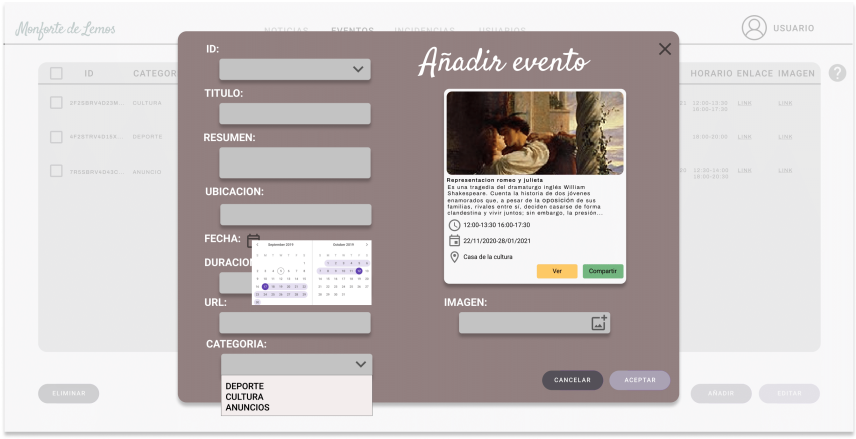


Figura A.6: Prototipo añadir eventos

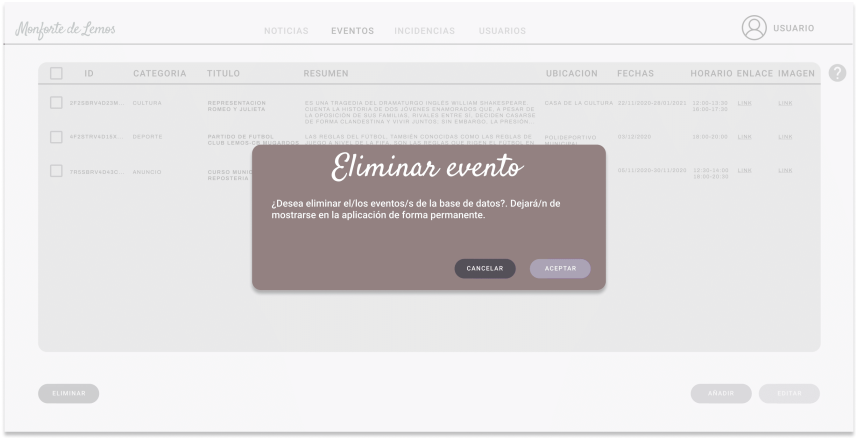


Figura A.7: Prototipo eliminación de eventos

A.4 Administración apartado Incidencias

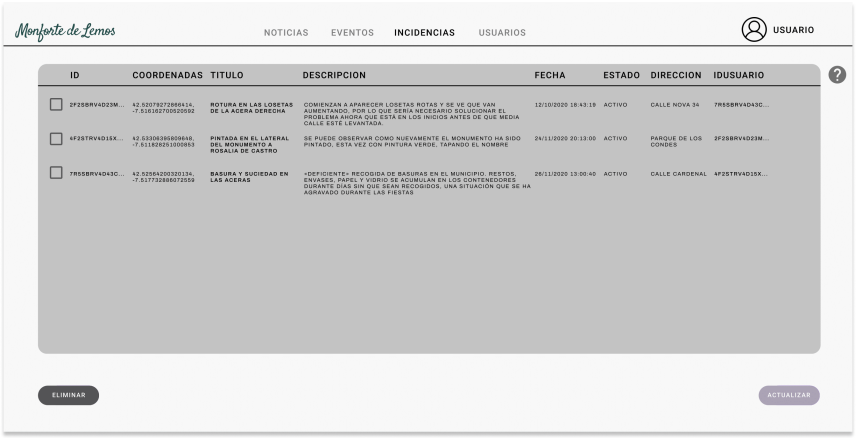


Figura A.8: Prototipo parrilla incidencias

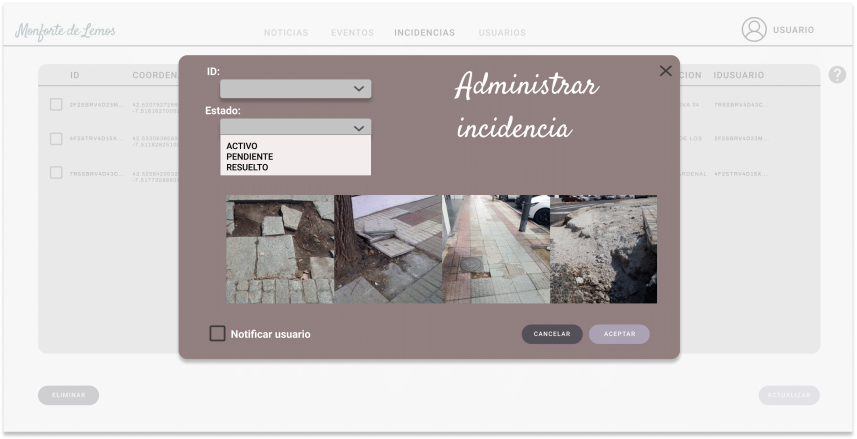


Figura A.9: Prototipo administrar incidencias

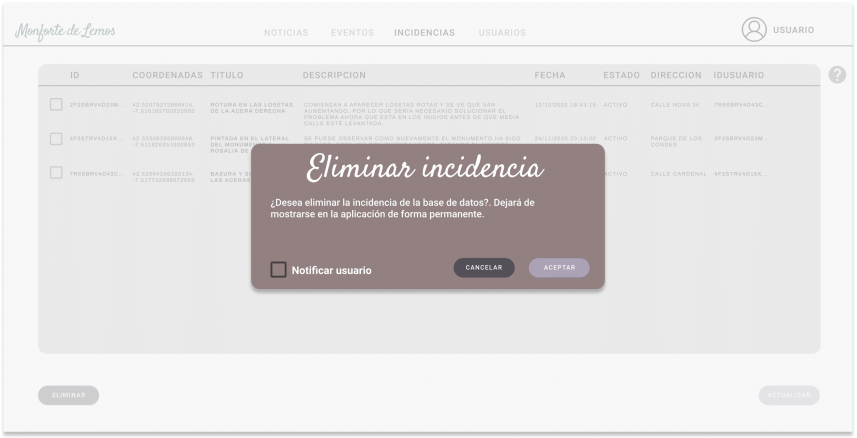


Figura A.10: Prototipo eliminación de incidencias

A.5 Administración apartado Usuarios

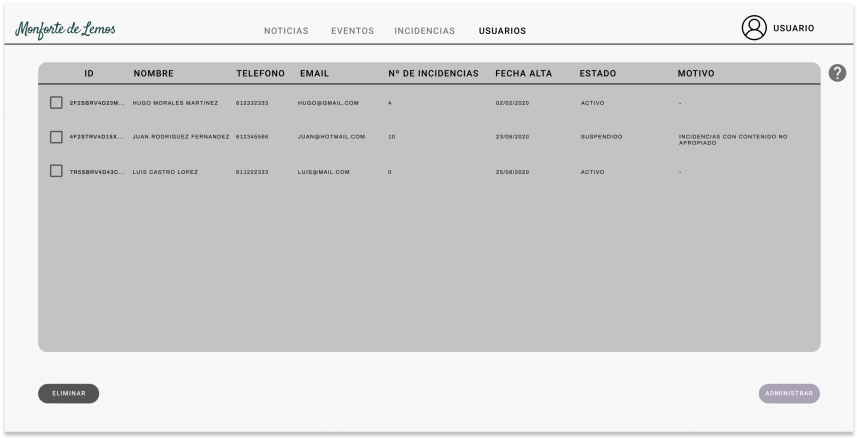


Figura A.11: Prototipo parrilla usuarios

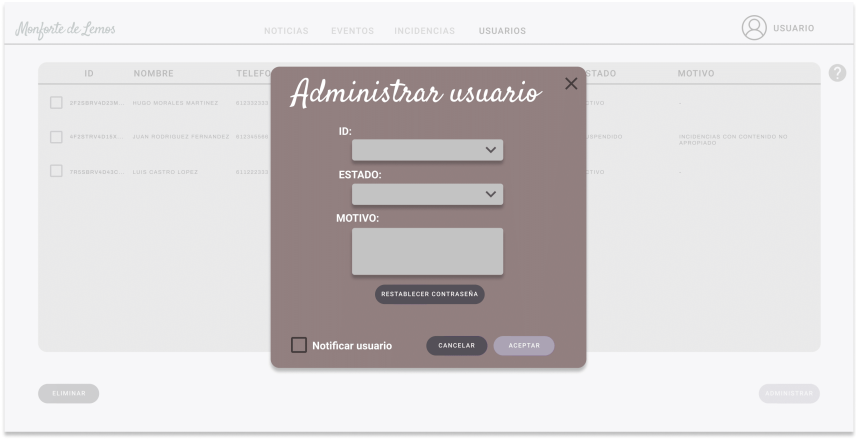


Figura A.12: Prototipo administrar usuarios

Figura A.13: Prototipo eliminación de usuarios

Lista de acrónimos

API *Application Programming Interface.*

CASE *Computer Aided Software Engineering*

CBSE *Component Based Software Engineering.*

GPS *Global Positioning System.*

HTTP *HyperText Transfer Protocol.*

JSON *JavaScript Object Notation.*

MVC *Model View Controller.*

MVVM *Model View ViewModel.*

NoSQL *Not Structured Query Language.*

RAM *Random Access Memory.*

SDK *Software Development Kit.*

SMS *Short Message Service.*

URL *Uniform Resource Locator.*

Glosario

AdMob Es un sistema de monetización de aplicaciones móviles. Se emplea para generar ingresos a través de la integración de publicidad orientada.

Card En el framework Flutter se corresponde con un widget cuya apariencia se relaciona con un panel con esquinas ligeramente redondeadas y una sombra de elevación. Pertenece a la guía Material design y se utiliza para representar información.

Container En el framework Flutter es un widget que combina el color, la posición y el tamaño de los widgets que contiene. Se utiliza para almacenar uno o más widgets, pudiendo modificar su presentación y colocación en la pantalla.

Drawer En el framework Flutter es un panel desplegable que se desliza horizontalmente desde el borde izquierdo de la pantalla, y proporciona un menú de navegación en la app.

Flatbutton En el framework Flutter es un widget consistente en un botón plano, formado por una etiqueta de texto que reacciona a los toques.

Framework Se trata de una estructura base empleada como inicio en la elaboración de un proyecto con unos objetivos específicos.

MockUps Prototipos o diseños de un producto software. Normalmente se representan sobre el dispositivo donde será utilizado el producto final. En algunos casos, estos entornos nos permiten simular el funcionamiento.

Scaffold En el framework Flutter es un widget que representa la disposición visual básica de la guía de Material Design. Proporciona la posibilidad de añadir menús laterales, alertas y desplegables.

Triggers Objetos que se almacenan en las bases de datos y que se ejecutan cuando sucede un determinado evento sobre las tablas a las que está asociado.

Bibliografía

[1] Empresa Estudio Alfa. [En línea]. Disponible en: https://estudioalfa.com/aplicaciones-moviles-apps-ayuntamientos

[2] Empresa Gestión Ayuntamiento. [En línea]. Disponible en: https://www.gestionayuntamiento.com

[3] Aplicación ayuntamiento de Cáceres. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.caceres.appmiciudad&gl=ES

[4] Aplicación Avisos Madrid. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=es.madrid.SGRSAMVANDCIU

[5] Herramienta Trello. [En línea]. Disponible en: https://trello.com/

[6] Salario medio Scrum Master en España. [En línea]. Disponible en: https://es.indeed.com/cmp/Between-Technology/salaries/scrum-master

[7] Salario medio analista software en España. [En línea]. Disponible en: https://es.indeed.com/cmp/Npr-Spain/salaries/analista-programador

[8] Salario medio diseñador UX en España. [En línea]. Disponible en: https://es.indeed.com/salaries/diseñador-web-Salaries

[9] Salario medio programador plataformas móviles en España. [En línea]. Disponible en: https://es.indeed.com/cmp/Viavox-Interactive,-S.l./salaries/analista-programador-ios

[10] Flutter. [En línea]. Disponible en: https://flutter.dev/docs

[11] Dart. [En línea]. Disponible en: https://dart.dev/

[12] Firebase. [En línea]. Disponible en: https://firebase.google.com

[13] NewsAPI. [En línea]. Disponible en: https://newsapi.org

[14] Google Maps API. [En línea]. Disponible en: https://cloud.google.com/maps-platform/maps?hl=es

[15] Figma. [En línea]. Disponible en: https://www.figma.com/

[16] MagicDraw. [En línea]. Disponible en: https://www.nomagic.com/products/magicdraw

[17] Android Studio. [En línea]. Disponible en: https://developer.android.com/studio

[18] Xcode. [En línea]. Disponible en: https://developer.apple.com/xcode/

[19] GitLab. [En línea]. Disponible en: https://about.gitlab.com

[20] Overleaf. [En línea]. Disponible en: https://es.overleaf.com/